MULTIPLE ATTRIBUTE DECISION MAKING METHODS: A COMPARATIVE STUDY

by

DANIEL K. McANARNEY

B.S. (Electrical Engg), Kansas State University, 1981

A MASTER'S THESIS

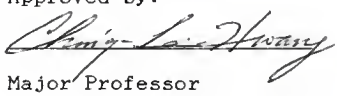
submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

College of Engineering
Department of Industrial Engineering
Kansas State University
Manhattan, Kansas

1987

Approved by:



Major Professor

LD
2668
.T4
HE
1987
M32
c. 2

A11207 309228

ACKNOWLEDGMENTS

I wish to express my gratitude to my major professor, Dr. C. L. Hwang. His guidance was invaluable in directing this work. A special thanks is extended to my committee members, Dr. Eddie Fowler, and Dr. Frank Tillman.

Dr. Brad Kramer was most helpful with his continual encouragement to greater diligence. Mrs Catheryn Long for her help in cutting through the universities red tape.

I am extremely grateful to my parents, Paul and Reva, whose support and encouragement are constant reminders to pursue excellence.

Finally, I would be remiss if I did not acknowledge the help and support of God, my creator. "A faithful friend is a sturdy shelter, he who finds one has found a treasure without price."

TABLE OF CONTENTS

ACKNOWLWDGEMENTS.....iii

LIST OF FIGURES.....vi

LIST OF TABLES.....vii

CHAPTER 1. Introduction.....1

1.1 Reasons for study.....2

1.2 Programming the algorithms.....3

1.3 MADM Terminology.....4

1.4 Summary.....5

CHAPTER 2. DOMINANCE AND EIGENVECTOR PAIRWISE COMPARISON

2.1 What is the DOMINANCE method?.....7

2.2 Ability of Dominance to Screen vs Number of Attributes.....8

2.3 Number of Expected Nondominated Solutions...8

2.5 What is new about this method?.....12

2.6 Selection of the top 4 to 6 attributes by the Eigenvector Method.....13

2.7 Necessary conditions for the selection of this method.....14

2.8 Operation of the attribute selection stage combined with dominance.....15

2.9 Results from sample problem.....17

CHAPTER 3. Multiple Level Attribute and Weight Generator and SAWM

3.1 Multi-level attribute and weight generation.20

3.2 Conditions affecting the selection of multi-level weight generation.....22

3.3 Results from a sample problem.....23

3.4 What is the SAWM method?.....24

3.5 Usage of the SAWM method.....27

3.6 Necessary conditions for using SAWM.....27

3.7 Sample problems using the SAWM method.....27

3.8 Using the SAWM method with HAWM.....29

CHAPTER 4 ELECTRE

4.1 What is the ELECTRE method.....30

4.2 ELECTRE's outranking relationship.....31

4.3 The ELECTRE methodology.....31

4.4 Sample problems using ELECTRE.....35

CHAPTER 5 TOPSIS

5.1 What is TOPSIS.....37

5.2 The positive and negative ideal solutions..38

5.3 The TOPSIS methodology.....38

5.4 Sample problems using TOPSIS.....41

5.5 Comparison of ELECTRE and TOPSIS.....42

CHAPTER 6. A COMPARISON OF RESULTS AMONG METHODS FOR LARGE PROBLEMS

6.1 The randomly generated problem.....44

6.2 TOPSIS vs SAWM.....45

6.3 TOPSIS and SAWM vs ELECTRE.....46

6.4 DOMINANCE.....47

6.5 Speed of execution.....54

6.6 Space considerations for the algorithms....56

6.7 Conclusions.....57

CHAPTER 7. CONCLUSIONS

7.1 Study review.....58

7.2 Overall results.....59

7.3 Considerations for future study.....60

REFERENCES.....62

APPENDIX A Expected number of dominated alternatives.....64

APPENDIX B Quantification of subjective attribute data.....66

APPENDIX C PASCAL Source Code.....68

APPENDIX D Supporting Code.....79

LIST OF TABLES

TABLE 1-1 Selection of an automobile.....2
TABLE 2-1 Demonstration of dominanted alternatives..8
TABLE 2-2 Expected number of nondominated alternatives.....11
TABLE 2-3 DM's pairwise comparisons.....18
TABLE 2-4 The DM's attribute weights.....18
TABLE 2-5 The selected attributes.....18
TABLE 2-6 The decision Matrix for shoe selection...19
TABLE 2-7 The non-dominated alternatives.....19
TABLE 3-1 The multi-level weights generated.....23
TABLE 3-2 Pairwise comparisons for the multi-level attributes.....24
TABLE 3-3 Attribute pairwise comparisons and weights.....28
TABLE 3-4 The alternatives' SAWM scores.....28
TABLE 3-5 Results from the fighter aircraft problem.....28
TABLE 3-6 Results of employment selection.....29
TABLE 4-1 Demonstration of the outranking relationship.....30
TABLE 5-1 Selection of the top two alternatives....30
TABLE 6-1 Decision matrix and results for a large problem.....48

TABLE 6-2 Batch file automating program bench marks.....54
TABLE 6-3 Benchmarks of MADM algorithms.....55

LIST OF FIGURES

FIGURE 2-1 Expected number of nondominated alternatives.....11
FIGURE 2-2 Flowchart of attribute selection-dominance.....16
FIGURE 3-1 Hierarchial employment selection structure.....21
FIGURE 3-2 Levels of the weight generator.....22

CHAPTER 1 Introduction

Decision making is a complex process. It involves judgment, information gathering, experience, and a myriad of other abilities. The study of decision making can cover several areas, from psychology to statistics to operations research. Each is concerned with a different area of the decision making process.

Decisions are often complex processes in themselves. Whether buying a car, choosing a job, or selecting a vacation site, each of the alternatives to be considered may have several competing attributes. Cars can be described in terms of cost, gas mileage, comfort, and status; jobs in terms of salary, advancement opportunities, and type of work; vacation sites in terms of travel time, climate, and so on.

1.1 Introduction

In this thesis we are considering the selection between alternatives for which we have data readily available; and the mathematical methods that are employed for this purpose. In Table 1-1 we have the data for three cars under consideration. Which car is the "best" for the decision maker in this problem? How can we make consistently reasonable choices on the basis of data such as this?

	Cost	Mileage	Comfort
Car #1	12,000.00	15.0	9
Car #2	6,000.00	28.0	5
Car #3	8,000.00	34.0	7

Table 1-1. Selection of an automobile.

These specific types of problems are referred to as multiple attribute decision making (MADM) problems; and it is the methods for solution of these problems with which we are concerned.

In 1957, Churchman, Ackoff, and Arnoff [13] first formally treated a MADM problem using the simple additive weighting method. Researchers have since developed other methods, many of these are included in reviews by MacCrimmon in 1968 [4] and 1973 [14], and Hwang and Yoon in 1981 [8].

1.1 Reasons for study

Our major interest in this study is in the practical application of MADM methods. Chapter 2 contains the examination of the dominance method combined with the eigenvector method of pairwise comparisons to enhance its usefulness. Dominance is most applicable when fewer attributes are present and the eigenvector method is used to reduce the initial number of attributes.

The development and implementation of a multiple level attribute and weight generator is discussed in chapter 3. This method consists of ordering the attributes into several logical levels and calculating the importance

(weight) of each attribute. The simple additive weighting method (SAWM) is combined with this method to select the preferred alternative.

Chapters 4 and 5 discuss underlying theory and subsequent applications of ELECTRE and TOPSIS. Their usefulness on larger problems is evaluated in Chapter 6.

The results of these methods are analyzed considering the goals of the decision maker (DM). Does the DM desire only the top alternative, an ordered list of alternatives, or the top ten percent of the list? Which methods give the most consistent results? Which methods are the most easily interpreted and implemented by the DM? If the DM cannot make use of the results, the methods are useless and should be discarded.

1.2 Programming of the algorithms

In our effort to evaluate these methods, computer programs were written in Turbo Pascal implementing each algorithm. Because a primary purpose of this thesis is to determine the usefulness of various MADM methods, care was taken not only to produce correct and readable results, but to facilitate problem input and storage. The goal was to produce programs which would be usable to persons unfamiliar with computers. Every attempt was made to produce bomb-proof code which would allow recovery from user errors without losing data already entered or causing

unnecessary user frustration. The code for each method is listed in Appendix C. All the remaining code, input/output, editing, etc. is listed in Appendix D.

Six programs were written: dominance, Technique for Order Preference by Similarity to Ideal Solution (TOPSIS), Elimination et Choice Translating Reality (ELECTRE), Simple Additive Weighting Method (SAWM), a multiple level attribute and weight generator (MULAG) and the eigenvector method used as a precursor to the dominance method. The first four, dominance, TOPSIS, ELECTRE, and SAWM require both alternative and attribute information. The next two, multiple level attribute weighting, and the eigenvector method require only attribute information. The dominance method needs only alternative information; no attribute weights are necessary.

1.3 MADM terminology

Describing MADM problems requires an understanding of the terminology used in the discussion of these problems. This section explains several of the terms.

Goal: Select a car. Now we can further define this goal by knowing the specific requirements for this car's use. Is this vehicle to be used to haul dynamite or the president? Different criteria are of greater or lesser importance in the performance of these two tasks. The DM

must have in mind a specific goal, a specific use or functional niche for the alternative he wishes to select.

Alternatives: Discrete options available to the decision maker. MADM is usually used to select between existing alternatives. Obviously, if one were selecting a car, one would desire the status of a Mercedes, the gas mileage of a Honda, the comfort of a Cadillac, and the road handling ability of a Lambourgini at the cost of a Subaru. But this is unrealistic. There is some alternative which is the best available balance of all factors, that may be satisfactory to the decision maker. This scenario also demonstrates the problem of conflicting alternatives.

Attributes: The properties of an alternative with which the decision maker is concerned.

Decision Matrix: An examination of table 1-1 readily demonstrates the ease of using the decision variables in a matrix format. The data defining the problem consists of m alternatives and n attributes arranged as a rectangular matrix. The set of alternatives with values collected for each attribute is called the decision matrix.

1.4 Summary

We will deal with several types of results the decision maker (DM) may desire. Does the DM want only the top alternative? Are the top 20 applicants from a pool of 200

desired for closer perusal? Does the DM only wish to order the list of alternatives from greatest to least?

We will be looking at three main ideas: how to determine the importance (weights) of the attributes used in the selection process, how to select the preferred alternatives from the given set of alternatives, and finally, how to make these methods more readily available to the decision maker.

CHAPTER 2 DOMINANCE AND EIGENVECTOR PAIRWISE COMPARISON

In this chapter we consider using the dominance method combined with an eigenvector pairwise comparison render improved solutions to MADM problems. The eigenvector comparison matrix is used to screen attributes so the dominance algorithm will produce a smaller subset of viable alternatives for further consideration. Then the DM could, if necessary use the information from the original set of attributes with the smaller subset of alternatives.

2.1 What is the Dominance method?

Dominance is used to screen a large set of multi-attributed alternatives into a smaller, more manageable number. A given alternative is dominated if there is another alternative that exceeds it in one or more attributes and equals it in all others. Dominance is used as an initial screening method on the set of alternatives before the final choice is made.

Dominance does not require the DM's preference information. Attributes need not be comparable. There are no special assumptions required of the attributes. Each alternative is compared across all attributes to determine if one alternative is dominated.

In Table 2-1, alternative A1 dominates alternative A2 because A1 meets A2 with attributes X1, X3, & X4, and exceeds it with attribute X2. There are no other dominated alternatives in the table.

	ATTRIBUTES			
ALTERNATIVE	X1	X2	X3	X4
A1	3	7	4	5
A2	3	6	4	5
A3	2	5	4	9
A4	2	6	3	8

Table 2-1. Demonstration of dominated alternatives.

2.2 Ability of Dominance to Screen vs Number of Attributes.

The dominance method will return fewer nondominated alternatives when fewer attributes are present. That is, more alternatives will be screened out or dominated when there are fewer attributes in the data. Consider again Table 2-1. If attribute X4 were the only attribute present, then alternative A3 would be the only nondominated alternative. Were attributes X4 and X2 the only attributes present, alternatives A2, A3, and A4 are nondominated. From this we can see that alternative A1 has the highest value for attribute X2, alternative A3 has the highest value for attribute X4, and alternative A4, although not the highest in either attribute X2 or X4 is also nondominated.

Thus, the number of attributes bears upon the number of nondominated alternatives expected.

2.3 Number of Expected Nondominated Solutions.

The expected number of nondominated alternatives from an initial set of m alternatives compared with respect to n attributes has been determined by Capline and Golding [1]. Consider the special case in which all the elements in the decision matrix are random numbers uniformly distributed over the range 0 to M. Attention is focused on the final (n^{th}) column. Arrange the rows so that the elements in the n^{th} column are in decreasing order of magnitude. By the randomness of the elements, the probability of an arbitrarily selected row being the r^{th} in the order is 1/m. Let p(m,n) be the probability that a row, arbitrarily chosen from the m rows (alternatives), is nondominated with respect to n attributes. Consider the r^{th} row. The ordering ensures that this row is not dominated by any row below it and also that it exceeds no row above it in the n^{th} attribute. Hence a necessary and sufficient condition for the r^{th} row to be nondominated is that it is nondominated among the first r candidates with respect to the first (n-1) attributes. Thus the probability of a row being the r^{th} and nondominated is p(r, n-1)/m. The probability of an arbitrarily selected row being nondominated is:

$$p(m, n) = \sum_{r=1}^{m} p(r, n-1)/m$$

$$= [p(m, n-1) + (m-1)\, p(m-1, n)]/m$$

Then the average number of nondominated alternatives, a(m,n), is:

$$a(m,n) = m * p(m, n)$$
$$= a(m, n-1)/m + a(m-1, n)$$

As a(m, 1) = a(1, n) = 1, a(m,n) can be calculated recursively. A good approximation of a(m,n) is given by:

$$a(m,n) = 1 + \ln m + (\ln m)^2/2! + \ldots + (\ln m)^{n-3}/(n-3)! + \gamma(\ln m)^{n-2}/(n-2)! + (\ln m)^{n-1}/(n-1)!$$

where γ is Euler's constant ($\approx$0.5772). Some typical results are shown in Figure 2-1. It indicates that the number of nondominated alternatives for a few attributes will be reduced greatly, whereas the number of nondominated alternatives for a large number of alternatives will still be very large.

In Appendix A the expected number of nondominated alternatives is calculated for several possible values of attributes and alternatives. Table 2-2 lists the expected number of nondominated alternatives for several initial quantities of attributes and alternatives. For example, with 100 alternatives run through the dominance method with 4 attributes, it is expected that there will be approximately 28 nondominated alternatives.

Number of initial alternatives ↓	Number of Attributes 2	3	4	5	6	7	8	9	10
1000	7	29	77	158	273	408	549	678	785
100	5	14	28	44	61	74	85	92	96
10	3	5	7	8	9	10	10	10	10

Table 2-2. Expected number of nondominated alternatives.

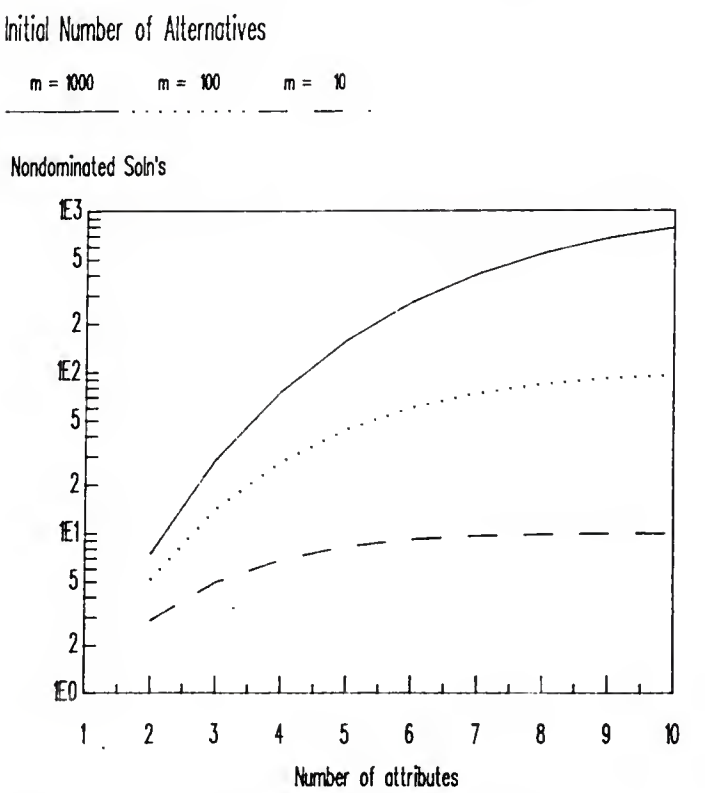


Figure 2-1 Expected number of nondominated solutions

2.4 Why do we use Dominance?

In MADM problems, the dominance method is used as a screening device to reduce time, labor and expense when selecting a few alternatives from a large pool of potential applicants. Consider a simple case of selecting 1000 new students at a private college from an applicant pool of 20,000 individuals. The DM wishes to discard 95 percent of the initial alternatives. There may be 15 to 20 criteria (attributes) that the DM wished to consider. The time involved to determine all the attributes for each alternative is enormous. The DM would thus be willing to consider a smaller number of the most important attributes to screen the initial population of 20,000 down to, say, 2000 or so alternatives. Then, this pool of 2000 could be selected on more carefully to determine the 1000 new students. This is the most important feature of the dominance method: screening a large number of alternatives into a more manageable number for the final selection.

2.5 What is new about this method?

To save the time, cost, and effort of screening a large number of candidates (alternatives) by the dominance method, only the most important 4 to 6 attributes are used. These top 4 to 6 attributes are determined using the eigenvector method. Then, the data for these attributes are collected for all alternatives. The data is then run

through the dominance method to determine the subset of nondominated alternatives.

2.6 Selection of the top 4 to 6 attributes by the Eigenvector Method

The eigenvector method is a method of scaling ratios using the principle eigenvector of the positive pairwise comparison matrix. The weights (the importance indexes) of the attributes are obtained [2], and the top 4 to 6 attributes are selected.

Only four to six attributes are selected to be used for screening with the dominance method because more attributes would not allow the alternatives to be screened easily.

The Decision Maker (DM) judges the relative importance of two attributes (criteria). The number of judgments, j, is: $j = n(n-1)/2$, where n is the number of attributes. Some inconsistencies from these judgments are allowed.

The four most important attributes, the attributes with the largest weights, are always kept. The fifth and sixth largest attributes are kept if each meets the following criteria: the weight of the attribute in question is over 50% of the average of the currently selected attributes, or the weight of the attribute in question is at least two thirds of the attribute with the least weight that has already been selected.

2.7 Necessary conditions for the selection of this method.

This METHOD is one of several MADM techniques. It has advantages and disadvantages when compared to other methods; and specific types of problems for which it is better suited to help the DM in his selection process. The following paragraphs highlight several aspects of the MADM problems that this METHOD is better suited for.

A major criteria to use in selecting this METHOD is the final result that the DM wishes. As this is a screening method, it would be useless if the DM desired to order the entire list of alternatives. This METHOD is useful when DM wishes to select only alternatives near the top, or the top alternative. For example, a personnel manager has received 200 applications for a position; this method would be useful in identifying the (approximately 20) candidates selected for interview.

There should be 100 or more alternatives and many attributes (8 to 25) in the data set. With fewer attributes, the number of alternatives removed from the initial set of alternatives would probably not justify the time and effort necessary to use this method.

The data need not be collected (or entered). A main advantage of this METHOD is in decreasing the amount of data to be collected.

2.8 Operation of the attribute selection stage combined with dominance.

* DM enters the attributes used to select among the alternatives.

* The DM does a pairwise comparison of the attributes.

* The attribute weights are calculated using the eigenvector method.

* The attributes are sorted in order of importance (by weight)

* The top four attributes are always selected.

* The next (fifth or sixth) most important attribute is selected if the weight of the attribute in question meets these criteria:

  * its weight is over 50% of the average of the currently selected set of attributes

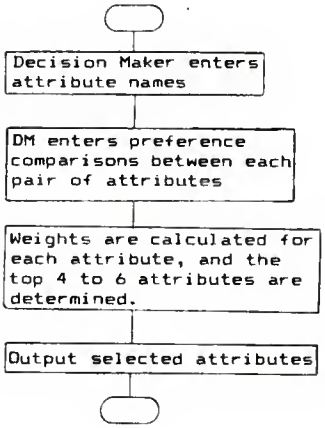
  * or its weight is at least two thirds of the weight of the attribute with the least weight that has already been selected.

* The DM collects the data. The attribute values are collected for the selected attributes for all alternatives.

* The DOMINANCE method is used to screen the list of alternatives.

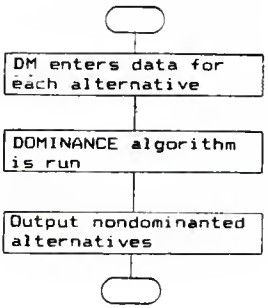
This is shown by the flowchart in Figure 2-2.

ATTRIBUTE SELECTION STAGE


Decision Maker enters attribute names
DM enters preference comparisons between each pair of attributes
Weights are calculated for each attribute, and the top 4 to 6 attributes are determined.
Output selected attributes

DATA COLLECTION STAGE

DOMINANCE STAGE



2.9 Results from sample problem.

In this problem, the decision maker was considering the purchase of a pair of shoes to be used for cycling.

The initial data was taken from the September/October 1984 issue of Bicycling. Seventeen shoes were listed, along with eighteen attributes for each shoe. Twelve of these attributes were of interest to the decision maker. These included weight, pedal grip, stability on pedal, protection from clips and straps, insole, midsole, shank, flex at ball of foot, heel cushioning, ventilation, padded collar, and padded tongue. These attributes were weighted using the first stage of this method, an eigenvector pairwise comparison. From this stage, five attributes were selected for input to the second stage, dominance. Table 2-3 contains the results of the DM's pairwise comparisons between each pair of attributes and Table 2-4 shows the resulting weights for each attribute. Table 2-5 lists the attributes selected for consideration using the dominance algorithm.

	weight	P_Grip	stable	clips	insole	midsole
weight	1.0	0.3	0.3	3.0	2.0	2.0
P_Grip	4.0	1.0	2.0	4.0	3.0	3.0
stable	3.0	0.5	1.0	3.0	5.0	5.0
clips	0.3	0.3	0.3	1.0	1.0	2.0
insole	0.5	0.3	0.2	1.0	1.0	1.0
midsole	0.5	0.3	0.2	0.5	1.0	1.0
shank	0.3	0.2	0.3	0.5	1.0	0.5
flex	2.0	0.5	0.5	3.0	3.0	3.0
cush'n	0.3	0.5	1.0	3.0	3.0	3.0
ventl'n	0.3	0.3	0.3	2.0	2.0	2.0
Collar	0.2	0.2	0.3	0.2	2.0	2.0
Tongue	0.2	0.2	0.3	0.2	2.0	2.0

	shank	flex	cush'n	ventl'n	Collar	Tongue
weight	3.0	0.5	3.0	3.0	5.0	5.0
P_Grip	5.0	2.0	2.0	4.0	5.0	5.0
stable	4.0	2.0	1.0	3.0	4.0	4.0
clips	2.0	0.3	0.3	0.5	5.0	5.0
insole	1.0	0.3	0.3	0.5	0.5	0.5
midsole	2.0	0.3	0.3	0.5	0.5	0.5
shank	1.0	0.2	0.3	0.5	0.5	0.5
flex	5.0	1.0	2.0	2.0	3.0	4.0
cush'n	4.0	0.5	1.0	2.0	4.0	4.0
ventl'n	2.0	0.5	0.5	1.0	3.0	3.0
Collar	2.0	0.3	0.3	0.3	1.0	1.0
Tongue	2.0	0.3	0.3	0.3	1.0	1.0

Table 2-3. The decision makers' pairwise comparisons.

weight	P_Grip	stable	clips	insole	midsole
0.11	0.20	0.16	0.06	0.04	0.04
shank	flex	cush'n	ventl'n	Collar	Tongue
0.03	0.12	0.11	0.07	0.04	0.04

Table 2-4. The decision makers' attribute weights.

weight	P_Grip	stable	flex	cush'n

Table 2-5. The selected attributes.

The attributes selected by the eigenvector method for consideration with the dominance method as shown in Table 2-5 were: weight, pedal grip, stability on pedal, flexibility, and heel cushioning. Weight is a cost criteria, the remainder are benefit criteria. These attributes were input to the dominance method with the data for the seventeen pairs of shoes. Table 2-6 contains a listing of the decision matrix for the seventeen shoes with the five selected attributes. The dominance method gave the result listed in Table 2-7.

	Weight	P_Grip	Stable
Avocet Mod 20	326.0	4.0	3.0
Avocet Mod 30	321.0	4.0	3.0
Bata Tourismo	297.0	4.0	3.0
Bata Sprintour	377.0	1.0	1.0
Brancale Touring #040	391.0	1.0	2.0
Duegi #116	270.0	1.0	3.0
Duegi #202	396.0	1.0	3.0
G. Bianchi #0115	337.0	3.0	2.0
Le Coq Sportif Cyclo	480.0	2.0	3.0
Le Coq Sportif Touring	422.0	2.0	3.0
Nike Velo	285.0	3.0	3.0
Patric Cyclo	338.0	3.0	3.0
Rivat Tergal 640	341.0	1.0	3.0
Rivat Vanoise Touring	415.0	1.0	3.0
Sidi Cycle Comfort	363.0	1.0	2.0
Vittoria Touring	289.0	1.0	3.0

	Flex	Cush'n
Avocet Mod 20	3.0	3.0
Avocet Mod 30	3.0	3.0
Bata Tourismo	2.0	1.0
Bata Sprintour	2.0	2.0
Brancale Touring #040	1.0	2.0
Duegi #116	1.0	1.0
Duegi #202	1.0	3.0
G. Bianchi #0115	3.0	1.0
Le Coq Sportif Cyclo	3.0	1.0
Le Coq Sportif Touring	3.0	1.0
Nike Velo	3.0	3.0
Patric Cyclo	2.0	2.0
Rivat Tergal 640	1.0	1.0
Rivat Vanoise Touring	1.0	1.0
Sidi Cycle Comfort	1.0	1.0
Vittoria Touring	2.0	1.0

Table 2-6. The decision matrix.

	Weight	P_Grip	Stable
Avocet Mod 30	321.0	4.0	3.0
Bata Tourismo	297.0	4.0	3.0
Duegi #116	270.0	1.0	3.0
Nike Velo	285.0	3.0	3.0

	Flex	Cush'n
Avocet Mod 30	3.0	3.0
Bata Tourismo	2.0	1.0
Duegi #116	1.0	1.0
Nike Velo	3.0	3.0

Table 2-7. The nondominated alternatives.

CHAPTER 3 Multiple level attribute and weight generation and SAWM

Multiple level attribute and weight generation and SAWM (simple additive weighting method) are two discrete types of MADM methods. Multiple level attribute and weight generation is used in the assessment of weights, the attributes of which have been described in a hierarchical manner. SAWM is used in determining which alternative is the preferred alternative from the available set. The two methods are described individually, and then considered as a combination.

3.1 Multiple level attribute and weight generation

Often, in MADM problems, the decision maker defines the attributes of concern in a hierarchical manner. There are several major criteria, each of which may be composed of sub-criteria. Consider an individual seeking employment who has received several job offers and is deciding between the alternatives. The candidate has the major criteria: type of work, geographic location, monetary compensation, and travel requirements. The major criteria of geographic location has the qualifier attributes of climate, proximity to relatives, and degree of urbanity. This multiple level set of attributes is shown in Figure 3-1.

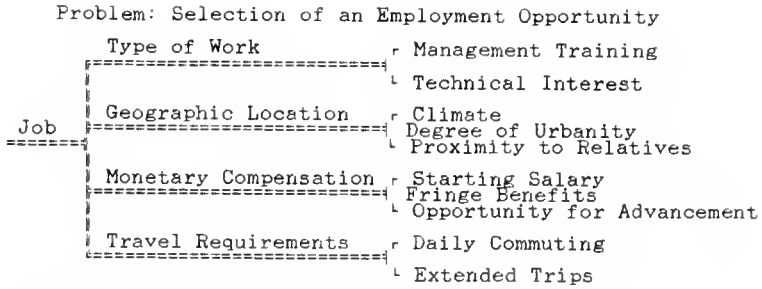


Figure 3-1. Multiple level employment selection structure.

It is possible to further break down an attribute. It would not be unreasonable to subdivide the criteria of fringe benefits into the criteria of retirement benefits, and insurance. In our work here, however, we will only deal with two levels of attributes.

The calculation of weights using this method is relatively simple and straightforward. Each level of the hierarchy is weighted using any existing weighting method which is acceptable to the decision maker. Our program allows either the eigenvector method or a direct entry of weights. In our employment selection example, there are five levels to be weighted. The first level consists of the four major criteria. The remaining four levels are the groups of sub-criteria beneath each major group (Figure 3-2).

```
Level 1: Employment Selection:
    Type of Work
    Geographic Location
    Monetary Compensation
    Travel Requirements

    Level 2.1: Type of Work:
        Management Training
        Technical Interest

    Level 2.2: Geographic Location:
        Climate
        Degree of Urbanity
        Proximity to Relatives

    Level 2.3: Monetary Compensation:
        Starting Salary
        Fringe Benefits
        Opportunity for Advancement

    Level 2.4: Travel Requirements:
        Daily Commuting
        Extended Trips
```

Figure 3-2. Levels of the weight generator.

After the individual levels have been weighted, the global weight for each leaf attribute is simply the product of its local weight and the weight of the attribute on the level directly above.

3.2 Conditions affecting the selection of this method

This method should be selected whenever there is an obvious grouping of one or more of the attributes. This will almost always be the case where ten or more attributes are present. Another case where a problem should be conformed to this method is when the attributes seem to be difficult to compare. How easily is climate compared to

starting salary? The decision maker should consider adding attributes to combine the existing attributes in a more orderly way.

3.3 Results from a sample problem.

The problem of selecting an employment opportunity is weighted using multiple level weight generation. The levels are weighted using the Pascal program implementing this method. The local weights are generated for each level, as shown in figure 3-2, and then multiplied to form the global weight for each leaf attribute. Table 3-1 shows the local and global weights for each attribute.

Problem Title: Job Selection

Attribute name	local weight	global weight
Type of Work 0.17		
Management Trng	0.25	0.04
Tech Interest	0.75	0.13
Geographic Locn 0.35		
Climate	0.12	0.04
Urbanity	0.27	0.09
Close to Home	0.61	0.21
Monetary Compns 0.28		
Starting Salary	0.57	0.16
Fringe Benefits	0.29	0.08
Optnty Advance	0.14	0.04
Travel Reqmnts 0.20		
Daily Commute	0.75	0.15
Extended Trips	0.25	0.05

Table 3-1. The multiple level weights generated.

Problem Title: Job Selection consistency: 0.46

name	weight	Type of	Geograp	Monetar	Travel
Type of Work	(0.17)	1.00	0.33	0.33	3.00
Geographic Locn	(0.35)	3.00	1.00	3.00	0.50
Monetary Compns	(0.28)	3.00	0.33	1.00	3.00
Travel Reqmnts	(0.20)	0.33	2.00	0.33	1.00

Attribute name: Type of Work consistency: -0.00

name	weight	Managem	Tech In
Management Trng	(0.25)	1.00	0.33
Tech Interest	(0.75)	3.00	1.00

Attribute name: Geographic Locn consistency: 0.04

name	weight	Climate	Urbanit	Close
Climate	(0.12)	1.00	0.33	0.25
Urbanity	(0.27)	3.00	1.00	0.33
Close to Home	(0.61)	4.00	3.00	1.00

Attribute name: Monetary Compns consistency: 0.07

name	weight	Startin	Fringe	Optnty
Starting Salary	(0.57)	1.00	3.00	3.00
Fringe Benefits	(0.29)	0.33	1.00	3.00
Optnty Advance	(0.14)	0.33	0.33	1.00

Attribute name: Travel Reqmnts consistency: -0.00

name	weight	Daily C	Extende
Daily Commute	(0.75)	1.00	3.00
Extended Trips	(0.25)	0.33	1.00

Table 3-2. Pairwise comparisons for the multiple level attributes.

3.4 What is the SAWM method?

SAWM is an acronym for Simple Additive Weighting Method. This is likely the best known and most widely used MADM method. This method is well summarized by MacCrimmon [4]. Churchman and Ackoff [5], and Klee [6] have a good basic discussion of the underlying considerations of SAWM.

SAWM consists of multiplying the attribute weights by the attribute value of each alternative and summing across the alternative. This produces a score for each alternative, the highest one being the most preferred alternative. In order not to unduly weight one attribute over the others (because of large data values) the decision matrix is scaled prior to summing the products of weights and attribute values to produce a numerically comparable scale.

This numerically comparable scale is formed for each attribute's values by doing a linear scale transformation with respect to the maximum value of the attribute, provided that the criteria (attributes) are defined as benefit criteria (the larger the x_j, the greater the benefit); thus the transformation is determined to be:

$$r_{ij} = \frac{x_{ij}}{x_i^{max}}$$

where $x_i^{max} = \max_i x_{ij}$, and for cost criteria:

$$r_{ij} = \frac{x_i^{min}}{x_{ij}}$$

where $x_i^{min} = \min_i x_{ij}$

The attributes are assigned weights by the decision maker which are used for coefficients to each alternative value corresponding to that attribute. The decision maker can then determine the score for each alternative by multiplying the scale rated attribute value (r_{ij}) with the weight assigned to this attribute and summing over all the attributes. The scores are computed for each alternative and the alternative with the highest score is the most favored alternative.

The DM assigns a set of importance weights to the attributes, $\underline{w} = \{ w_1, w_2, \ldots, w_n\}$. Then the most preferred attribute, A^*, is selected such that:

$$A^* = \{A_i \mid \max_i \sum_{j=1}^{n} w_j * x_{ij} / \sum_{j=1}^{n} w_j \}$$

where x_{ij} is the outcome of the i^{th} alternative about the j^{th} attribute with a numerically comparable scale. The weights are normalized such that they sum to one,

$$\sum_{j=1}^{n} w_j = 1$$

the equation reduces to:

$$A^* = \{A_i \mid \max_i \sum_{j=1}^{n} wj_* x_{ij} / \sum_{j=1}^{n} w_j \}$$

3.5 Usage of the SAWM method.

The SAWM method gives cardinal values for each alternative. The most preferred alternative has the highest value, less preferred have successively lower values. This allows the DM to extract several useful results: selection of the most preferred alternative, selection of the top alternatives, or simply ordering the utility of all alternatives.

3.6 Necessary conditions for using SAWM

SAWM requires that the attribute values be numerical and comparable across attributes. The attributes can be converted simply to a numerical scale (Appendix B). The attributes must be comparable because they are combined across attributes. This is a weak point in SAWM; often attributes cannot be considered separately and added together because of complementarities between attributes. A "high" value for one attribute must receive approximately the same numerical values as "high" values of other attributes.

3.7 Sample problems using the SAWM method.

The Decision Maker has a choice between three automobiles. The criteria used to discriminate between the alternatives are: cost, mileage, and comfort. Table 3-3 contains the decision matrix. All attributes are weighted

equally. The SAWM algorithm produced the results shown in Table 3-4.

	Cost	Mileage	Comfort
Car #1	12000.00	15.00	9.00
Car #2	6000.00	28.00	5.00
Car #3	8000.00	34.00	7.00

Table 3-3. Attribute pairwise comparisons and weights

Car #1	0.65
Car #2	0.79
Car #3	0.84

Table 3-4. The alternatives SAWM scores.

The second problem chosen for solution using the SAWM method is a fighter aircraft problem. Table 3-5 contains the decision matrix, the attribute weights, and the SAWM scores for this problem.

The Decision Matrix:

	speed	payload	ceiling	cost	manuver	rely
a1	2.00	1500.00	20000.00	5.50	5.00	9.00
a2	2.50	2700.00	18000.00	6.50	3.00	5.00
a3	1.80	2000.00	21000.00	4.50	7.00	7.00
a4	2.20	1800.00	20000.00	5.00	5.00	5.00

The DM subjective weights:

speed	payload	ceiling	cost	manuver	rely
0.20	0.10	0.10	0.10	0.20	0.30

Alternative selection values:

a1	0.84
a2	0.71
a3	0.85
a4	0.74

Table 3-5. Results from the aircraft problem.

3.8 Using the SAWM method with HAWM

The decision matrix containing the attribute values for each of the leaf variables in the multiple level weighting problem is shown in Table 3-6

The Decision Matrix

	Manage	Techni	Climat	Degree	Proxim
a1	7.0	4.0	7.0	2.5	0.0
a2	8.5	6.0	7.0	2.5	0.0
a3	7.5	7.5	8.5	1.0	5.0
a4	8.0	8.5	6.0	15.0	1.0

	Starti	Fringe	Opport	Daily	Extend
a1	8.1	1.22	15.0	0.50	0.0
a2	8.3	1.55	10.0	1.00	5.0
a3	8.7	1.57	11.0	0.25	0.0
a4	8.5	1.27	11.0	1.25	2.0

The DM subjective weights

Manage	Techni	Climat	Degree	Proxim
0.04	0.13	0.04	0.09	0.21
Starti	**Fringe**	**Opport**	**Daily**	**Extend**
0.16	0.08	0.04	0.15	0.05

Alternative selection values

a1	0.46
a2	0.61
a3	0.71
a4	0.76

Table 3-6. Results of employment selection.

CHAPTER 4 ELECTRE

4.1 What is the ELECTRE method

ELECTRE is a Multiple Attribute Decision Making (MADM) method. Often in MADM problems there are no completely dominated alternatives. ELECTRE is based on a concept of "ELECTREcally" comparing the alternatives to determine dominated alternatives. The attribute data is used in this method determine the degree of dominance between alternatives. This is done using a pairwise comparison of the alternatives with respect to the weights of each attribute.

In Table 4-1, there are no dominated alternatives. However, from an inspection of Table 4-1, alternative A1 nearly dominates alternative A2, and one would expect that unless attribute X1 were very important, alternative A1 would always be selected over alternative A2. Alternative A1 exceeds alternative A2 in attributes X2, X3, and X4; and there is little difference between them in attribute A1. This is the basis of ELECTRE's "outranking relationship".

	ATTRIBUTES			
ALTERNATIVE	X1	X2	X3	X4
A1	3	7	5	6
A2	3.1	6	4	5
A3	2	5	4	9
A4	2	6	3	8

Table 4-1. Demonstration of the outranking relationship.

4.2 ELECTRE's outranking relationship

An alternative A_k outranks an alternative A_j when the available data are such that the DM accepts the proposition that "A_k is at least as good as A_j". The meaning of "at least as good" is determined in ELECTRE by examining two subsets of the attributes. These subsets are called the concordance and discordance sets and are evaluated for each pair of alternatives. The concordance set consists all the attributes of alternative A_k that are preferred to alternative A_j, and likewise, the discordance set contains all the attributes for which A_j is preferred to A_k. Alternative A_j is outranked (ELECTREcally dominated) by alternative A_j when a sufficiently large (important) subset of the attributes fall in the concordance set, and a sufficiently small (unimportant) subset falls into the discordance set.

4.3 The ELECTRE methodology.

Step 1. Calculate the normalized decision matrix. Normalizes the decision matrix according to the formula:

$$a_{ij} = a_{ij} / \sqrt{\left[\sum_{i=1}^{m} (a_{ij})2 \right]}$$

Step 2. Calculate the weighted normalized decision matrix.

$$a_{ij} = a_{ij} * \left[w_i / \sum_{i=1}^{m} w_i \right]$$

Where the weights $\{w_1, w_2, \ldots w_m\}$ have been normalized to sum to one, the equation reduces to:

$$a_{ij} = a_{ij} * w_i$$

Step 3. Calculate the concordance matrix: Each item of the matrix is produced by a pairwise comparison of two alternatives. The value of the matrix item c_{kl} is equal to the sum of the (normalized) weights of the attributes where alternative A_k is preferred to alternative A_j.

$$c_{kl} = \sum_{j \in C_{kl}} w_j$$

where $C_{kl} = \{j \mid x_{kj} \geq x_{lj}\}$. This concordance index indicates the relative importance of A_k with respect to A_l. c_{kl} is in the range $0 \leq c_{kl} \leq 1$. The higher the value of c_{kl} the more preferred A_i is to A_k. The concordance index is calculated for all pairs of alternatives to form the concordance matrix, C.

$$C = \begin{bmatrix} --- & c12 & c13 & \ldots & c1m \\ c21 & --- & c23 & & c2m \\ & & & & \\ & & & & \\ cm1 & cm2 & & cm(m-1) & --- \end{bmatrix}$$

Generally, in the matrix C, $c_{kl} + c_{lk} = 1$. This is true for comparisons between alternatives where they have no identical attribute values. If two alternatives have the same value for one or more attributes then $c_{kl} + c_{lk} > 1$. For ease of later calculations, the diagonal of the matrix is set equal to zero.

Step 4. Calculate the discordance matrix: The concordance values indicate the relative dominance of Alternative Ak over alternative Al by using the sum of the weights of the attributes where Ak equals or exceeds Al. The discordance matrix uses, not the sum of the corresponding weights, but a measure which includes the difference of alternative values.

$$d_{kl} = \frac{\max_{j \in D_{kl}} | v_{kj} - v_{lj} |}{\max_{j \in J} | v_{kj} - v_{lj} |}$$

where $D_{kl} = \{j | x_{kj} \geq x_{lj}\}$. For ease of later calculations, the diagonal of the matrix is set equal to zero.

$$D = \begin{bmatrix} --- & d12 & d13 & \cdots & d1m \\ d21 & --- & d23 & & d2m \\ & & & & \\ & & & & \\ dm1 & dm2 & & dm(m-1) & --- \end{bmatrix}$$

Step 5. Determine the concordance threshold: This threshold is calculated as a step in determining if alternative A_k dominates alternative A_l. This threshold value can be given by the decision maker or determined from the problem. The concordance threshold selected such that it is contained in the interval [0,1]. Alternative A_k will have a chance of dominating A_l only if its concordance index c_{kl} exceeds the concordance threshold, c_{bar}.

$$c_{kl} \geq c_{bar}$$

A good initial value for the threshold value is the average concordance index:

$$c_{bar} = \sum_{k=1}^{m} \sum_{l=1}^{m} [\, c_{kl} \,/\, m(m-1) \,]$$

Step 6. Determine the discordance threshold.: The discordance threshold is calculated like the concordance threshold and is contained in the interval [0,1]. Alternative A_k will have a chance of dominating A_l only if its discordance index d_{kl} does not exceed the discordance threshold, d_{bar}.

$$d_{kl} \geq d_{bar}$$

Again, a good initial value for the threshold value is the average discordance index:

$$d_{bar} = \sum_{k=1}^{m} \sum_{l=1}^{m} [\, d_{kl} \,/\, m(m-1) \,]$$

Step 7. Determine the outranking matrix (or aggregate dominance matrix): This matrix contains the values of the outranking relationship for all pairs of alternatives. The outranking relation is defined as a binary adjacent matrix: $O = (o_{kl} \mid k,l = 1,2,\ldots,m)$. Where

$$okl = \begin{cases} 1 \text{ if } ckl \geq cbar \text{ and } dkl \leq dbar \\ 0 \text{ otherwise} \end{cases}$$

4.4 Sample problems solved with the ELECTRE algorithm:

Problem I: Cycling shoe selection.

In this problem, we are using the nondominated alternatives from the cycling selection problem we solved in Chapter 2, Dominance.

The Decision Matrix

	Weight	P_Grip	Stable	Flex	Cush'n
Avocet Mod 30	321.000	4.000	3.000	3.000	3.000
Bata Tourismo	297.000	4.000	3.000	2.000	1.000
Duegi #116	270.000	1.000	3.000	1.000	1.000
Nike Velo	285.000	3.000	3.000	3.000	3.000

The weights from chapter 2 are used here, but as only the top 5 attributes are used, they have been normalized to sum to one.

The DM subjective weights:

Weight	P_Grip	Stable	Flex	Cush'n
0.157	0.286	0.229	0.171	0.157

The concordance threshold: 0.475
The discordance threshold: 0.595

The concordance dominance matrix

	Avocet	Bata T	Duegi	Nike V
Avocet Mod 30	0.000	0.590	0.590	0.590
Bata Tourismo	0.470	0.000	0.590	0.360
Duegi #116	0.270	0.380	0.000	0.270
Nike Velo	0.500	0.500	0.590	0.000

The discordance dominance matrix

	Avocet	Bata T	Duegi	Nike V
Avocet Mod 30	0.000	0.091	0.103	0.218
Bata Tourismo	1.000	0.000	0.055	1.000
Duegi #116	1.000	1.000	0.000	1.000
Nike Velo	1.000	0.627	0.045	0.000

The aggregate dominance matrix

	Avocet	Bata T	Duegi	Nike V
Avocet Mod 30	0.000	1.000	1.000	1.000
Bata Tourismo	0.000	0.000	1.000	0.000
Duegi #116	0.000	0.000	0.000	0.000
Nike Velo	0.000	0.000	1.000	0.000

The aggregate dominance matrix displays the following outranking relationships: Avocet → Bata, Avocet → Duegi, Avocet → Nike, Bata → Duegi, and Nike → Duegi; Giving the conclusion that the Avocet shoes are the only non (ELECTREcally) dominated solutions.

CHAPTER 5 TOPSIS

5.1 What is TOPSIS

TOPSIS is an algorithm used in Multiple Attribute Decision Making (MADM). TOPSIS is an acronym for Technique for Order Preference by Similarity to Ideal Solution. This method was originally developed by Hwang and Yoon [7, 8]. TOPSIS is based on the concept that the DM would select the alternative that is as close as possible to an ideal alternative and as far as possible from a negative ideal alternative. Topsis gives a value between 0 and 1 for each alternative indicating that alternative's relative closeness to the ideal solution.

TOPSIS can easily be used to find the most preferred attribute, but it also gives information not presented by either DOMINANCE or ELECTRE. Consider Table 5-1. Attributes X1-X4 are weighted equally. If the DM is only looking for the top alternative, he obviously selects A1. However, if the top two alternatives are to be selected how will the second alternative be chosen? A1 dominates A2, and ELECTREcally dominates A2, A3, and A4. And neither ELECTRE nor pure DOMINANCE indicate any preference between A2, A3, or A4.

TOPSIS provides an simple, yet powerful method of determining absolute preferences throughout the overall list, where DOMINANCE and ELECTRE give preferences only relative to each individual pair of alternatives.

	ATTRIBUTES			
ALTERNATIVE	X1	X2	X3	X4
A1	8	9	7	9
A2	7	9	7	8
A3	7	8	8	7
A4	6	8	8	8

Table 5-1. Selection of the top two alternatives

5.2 The positive and negative ideal solutions

The TOPSIS method constructs two standard alternatives by locating the best attribute values from the existing set of alternatives and combining them to form positive ideal solution. The negative ideal solution is formed by combining the worst attribute values from the decision matrix. These two ideal alternatives are used to as a comparison standard for each of the alternatives in the set. This is similar to the the concordance and discordance concepts in the ELECTRE method. The main difference being that the ELECTRE method has no standard alternative; each existing alternative was compared to all others.

5.3 The TOPSIS methodology

TOPSIS considers the distance to the ideal and negative ideal solution simultaneously by taking the relative closeness to the ideal solution. This method yields a

cardinal preference order as the solution to the MADM problem.

The TOPSIS decision matrix contains m alternatives and n attributes, and has the form:

$$D = \begin{array}{c} \\ A1 \\ A2 \\ \\ Ai \\ \\ Am \end{array} \begin{array}{c} \begin{array}{cccccccc} x1 & x2 & x3 & & xj & & xn \end{array} \\ \begin{bmatrix} x11 & x12 & x13 & \dots & x1j & \dots & x1n \\ x21 & x22 & x23 & \dots & x2j & \dots & x2n \\ \vdots & \vdots & & & & & \\ xi1 & xi2 & xi3 & \dots & xij & \dots & xin \\ \vdots & \vdots & & & & & \\ xm1 & xm2 & xm3 & \dots & xmj & \dots & xmn \end{bmatrix} \end{array}$$

where: A_i is the i^{th} alternative considered by the decision maker, x_j is the j^{th} attribute, and x_{ij} is the attribute value for the i^{th} alternative at the j^{th} attribute.

Step 1. Calculate the normalized decision matrix. Normalizes the decision matrix according to the formula:

$$r_{ij} = x_{ij} / \sqrt{\left[\sum_{i=1}^{m} (x_{ij})2 \right]}$$

forming a normalized decision matrix, R.

Step 2. Calculate the weighted normalized decision matrix. The weight assigned to each attribute is multiplied by each column of the R matrix, (the attribute values of each alternative in the decision matrix)

$$v_{ij} = r_{ij} * \left[w_i / \sum_{i=1}^{m} w_i \right]$$

Where the weights $\{w_1, w_2, \dots w_m\}$ have been normalized to sum to one, the equation reduces to:

$$a_{ij} = a_{ij} * w_i$$

Step 3. Determine the positive and negative ideal solutions: This step consists of constructing the standard alternatives: one of which is a composite of the best attribute values, named A^*. And the other which is a composite of the worst attribute values (A^-). With benefit criteria, the best attribute values are the maximum values; and with cost criteria, the best attribute values are minimum values. These standard alternatives are defined as:

$$A^* = \{(\max_i v_{ij} \mid j \in J), (\min_i v_{ij} \mid j \in J^+) \mid i=1,2,\ldots,m\}$$

$$= \{v_1^*, v_2^*, v_3^*, \ldots, v_j^*, \ldots, v_n^*\}$$

$$A^- = \{(\min_i v_{ij} \mid j \in J), (\max_i v_{ij} \mid j \in J^+) \mid i=1,2,\ldots,m\}$$

$$= \{v_1^-, v_2^-, v_3^-, \ldots, v_j^-, \ldots, v_n^-\}$$

where J is the set of all benefit criteria (attributes) and J^- is the set of all cost criteria.

Step 4. Calculate the separation measures. The separation of each alternative from the standard alternative is defined as the n-dimensional Euclidean distance. These separation measures are given by the formula:

$$S_{i*} = \sqrt{\left[\sum_{j=1}^{n} (v_{ij} - v_j^*)^2\right]}, \quad i = 1, 2, \ldots, m$$

$$S_{i-} = \sqrt{\left[\sum_{j=1}^{n} (v_{ij} - v_j^-)^2\right]}, \quad i = 1, 2, \ldots, m$$

Step 5. Calculate the relative closeness to the ideal solution. Recall that TOPSIS uses not only the distance of the alternative from the ideal solution, but a combination of the alternatives relative closeness to both the ideal and negative ideal solution. The relative closeness of alternative A_i with respect to the positive ideal solution, A^* is defined as:

$$C_{i*} = S_{i-} / (S_{i*} + S_{i-}), \quad i = 1, 2, \ldots m$$

From this we can see that C_{i*} is in the range from 0 to 1, and the closer an alternative is to the ideal solution, the closer C_{i*} approaches 1. This allows us to order the preference of the alternatives, A_i, by ordering the closeness measures of the alternatives, C_{i*}.

5.4 Sample problems using TOPSIS

In this problem, we are using the nondominated alternatives from the cycling selection problem we solved in Chapter 2, Dominance.

The Decision Matrix

	Weight	P_Grip	Stable	Flex	Cush'n
Avocet Mod 30	321.000	4.000	3.000	3.000	3.000
Bata Tourismo	297.000	4.000	3.000	2.000	1.000
Duegi #116	270.000	1.000	3.000	1.000	1.000
Nike Velo	285.000	3.000	3.000	3.000	3.000

The weights from chapter 2 are used here, but as only the top 5 attributes are used, they have been normalized to sum to one.

The DM subjective weights:

Weight	P_Grip	Stable	Flex	Cush'n
0.200	0.200	0.200	0.200	0.200

Alternative selection values

Avocet Mod 30	0.898
Bata Tourismo	0.507
Duegi #116	0.102
Nike Velo	0.815

Rank Order of alternatives

Avocet Mod 30	0.898
Nike Velo	0.815
Bata Tourismo	0.507
Duegi #116	0.102

5.5 Comparison of ELECTRE and TOPSIS

Both TOPSIS and ELECTRE are used for the same type of MADM problems, problems containing comparable, numerical attributes, where all the attributes are relevant to the decision and the DM wishes to find near optimal alternatives. They can be seen to encompass the same "type" of measurements to determine optimality. TOPSIS compares each alternative to the ideal and negative-ideal solutions; similarly, ELECTRE determines the concordance and discordance values, comparing each alternative with all other alternatives instead of an ideal alternative.

There are two areas where TOPSIS seems to be the better method. First, TOPSIS gives an cardinally ordered list as the result. This is often preferred to ELECTRE's outranking relationship in that it gives the DM an idea of how much an alternative is preferred to another. ELECTRE tells only which of two alternatives is more preferred, if

any, not how much it is preferred. Now when the DM considers all the outranking relationships more information can be discerned, but again, not as much or as readily as TOPSIS.

Second, TOPSIS requires less computer time to calculate and less memory to store intermediate and final results. Since ELECTRE compares each alternative to all others, a large pairwise comparison matrix is required for each of the concordance, discordance, and aggregate dominance matrices. This would not be a hindrance with MADM problems with fewer than approximately 50 alternatives, but could be very constrictive with, say, 1000 alternatives. This would require approximately 6 megabytes, assuming a 1000 by 1000 array of 6-byte reals, for each of the three matrices that ELECTRE needs.

ELECTRE and TOPSIS do give similar results for the same dataset. SAWM also give similar results. This is to be expected since the data and the goal are the same for each method. Chapter 6 contains the results of these three methods run on the same dataset, and some considerations of the differences in their results.

CHAPTER 6 Comparison of results among methods for large problems

We began this to better understand the interaction between methods, the compatibility of methods, and the suitability of these methods for larger problems. Our method consisted of generating a random problem containing 266 alternatives and 5 attributes. The attributes were all weighted similarly. The criteria we considered were: speed, storage requirements, consistency of results among and within each method.

6.1 The randomly generated problem:

A decision matrix of 266 alternatives each containing five attribute values was built using a random number generator. The attribute values were integers between 1 and 9. The decision matrix is shown in Table 6-1.

In an effort to save space on this problem and make the results more comparable, the results of all four methods (SAWM, TOPSIS, ELECTRE, and dominance) are displayed side by side, along with the decision matrix. The alternatives are sorted their TOPSIS ranking. This information is shown in figure 6-1. Non-dominated alternatives are indicated by a "*" immediately after the alternative name. It was not viable to print out the entire ELECTRE concordance discordance, and aggregate dominance matrices (nor were

they even directly calculated). The information in the last two columns indicate the number of alternatives that alternative i dominates (ELECTREcally) and the number of alternatives that dominate alternative i.

6.2 TOPSIS vs SAWM

From an inspection of the rankings of TOPSIS and SAWM, we can conclude that these methods give quite similar results. The results are quite consistent for the first ten to fifteen items. The first fifty items vary in their rankings by less than 15 places. Farther down the list, more variation is seen in the results between these two methods. There is a good correlation between these results through the entire list, but there are more alternatives that are ranked 20 or 30 places apart in the middle portion of the list. A closer inspection of the first fifty items will give us a better understanding of the practical criteria that lead to high rankings by one of these two methods, but not the other. Consider alternative A30. TOPSIS ranks this 31st., SAWM 17th with attribute values of 7, 7, 1, 9, and 9, this leads us to project that SAWM values an peaks (the two nines) and doesn't consider the lower attributes as much (the 1). This is shown well by alternative A138 which TOPSIS ranks 18th and SAWM 35th. Having attribute values of 7, 7, 7, 5, and 6, clearly

indicates TOPSIS' greater consideration of a high average with no severely low attribute values.

The reason this occurs is TOPSIS considers not only the ideal solution, but also the negative ideal solution. SAWM considers only closeness to the ideal solution. The SAWM solution is generally not acceptable except at the very top of the list. If all the attributes are included in the problem, they all should be considered. These two methods, TOPSIS and SAWM are correlated to the conjunctive and disjunctive methods. The basic concept of the conjunctive method is that each alternative must meet certain levels of acceptability with each alternative; if a basketball player is the best shooter in the world, but the worst dribbler, he would not be useful to the team. The disjunctive method is one in which each alternative is evaluated on its best attribute; a football player who is an excellent kicker, yet a very poor blocker is still an asset to the team. From this we would suggest that TOPSIS is the better method.

6.3 TOPSIS and SAWM vs ELECTRE

ELECTRE shows there are two alternatives which are not outranked by any others, and these two alternatives correspond to the first two selections of both TOPSIS and SAWM. Like the increasing variation between SAWM and TOPSIS, ELECTRE has good agreement near the top of the

list, but then shows greater and greater variation. One should also mention that the information ELECTRE presents is not as useful as the information given by TOPSIS or SAWM. Obviously the third ranking alternative (A143) is ELECTREcally dominated by A53 or A60, but which of the first two doesn't dominate A143, the farther down the ELECTRE list one goes the more and more interaction there is among alternatives. To be fair, if one printed the aggregate dominance matrix, this information would be present, but the matrix is hard to read and the various outranking relationships tell little after the first few alternatives.

6.4 DOMINANCE

It is our opinion that the dominance method is useless in determining preferred alternatives in MADM problems. Four of the top ten items selected by TOPSIS are dominated, as are four of ten in ELECTRE, and 3 of ten in SAWM. Of the top 20, eleven are dominated in TOPSIS, ten in SAWM, and nine in ELECTRE.

Decision Matrix

		x1	x2	x3	x4	x5	TOPSIS score	TOPSIS rank	SAWM score	SAWM rank	ELECTRE ai>n	ELECTRE n>ai
a53	*	9	6	9	9	8	0.84	1	0.91	1	262	0
a60	*	6	9	8	9	8	0.83	2	0.89	2	261	0
a143		6	9	8	8	8	0.81	3	0.87	4	259	1
a44	*	9	6	9	6	9	0.79	4	0.87	3	260	1
a213	*	9	7	7	6	8	0.77	5	0.82	6	246	2
a42		8	6	8	6	7	0.74	6	0.78	8	232	5
a203		6	8	8	9	5	0.73	7	0.80	7	243	3
a257	*	4	9	7	8	9	0.73	8	0.82	5	239	4
a54		6	6	7	7	8	0.72	9	0.76	14	234	7
a111	*	9	7	8	4	7	0.70	10	0.78	10	219	7
a187		6	6	9	5	8	0.69	11	0.76	13	222	9
a181	*	4	8	9	9	5	0.69	12	0.78	11	232	7
a85		5	5	9	9	6	0.68	13	0.76	16	221	7
a106	*	9	9	5	4	8	0.68	14	0.78	9	218	6
a200		9	3	7	7	8	0.68	15	0.76	15	186	7
a41		6	9	6	6	6	0.68	16	0.73	24	219	7
a182		7	4	7	6	9	0.67	17	0.73	20	197	10
a138		7	7	7	5	6	0.67	18	0.71	35	197	15
a79		5	8	6	7	6	0.66	19	0.71	34	205	13
a19	*	8	7	5	4	9	0.66	20	0.73	23	197	9
a50	*	7	9	3	7	7	0.65	21	0.73	21	183	9
a253		2	6	9	9	8	0.65	22	0.76	12	181	10
a247	*	6	7	3	8	9	0.65	23	0.73	22	186	9
a8		4	8	4	9	8	0.65	24	0.73	19	199	13
a66		4	5	8	9	6	0.65	25	0.71	33	188	11
a65		3	7	6	9	7	0.64	26	0.71	32	182	16
a219		4	8	4	8	8	0.63	27	0.71	31	183	16
a37		4	8	4	7	9	0.63	28	0.71	27	183	13
a83	*	9	8	2	9	5	0.63	29	0.73	18	180	11
a265		9	5	6	3	9	0.63	30	0.71	28	174	16
a30	*	7	7	1	9	9	0.62	31	0.73	17	146	12
a9	*	6	8	2	7	9	0.62	32	0.71	30	159	13
a209		4	8	4	8	7	0.62	33	0.69	36	170	24
a13	*	6	6	2	9	9	0.62	34	0.71	25	156	17
a170	*	8	9	5	5	4	0.62	35	0.69	43	178	15
a122	*	5	9	9	2	7	0.61	36	0.71	26	168	13
a139		9	2	4	8	8	0.61	37	0.69	38	143	15
a263		7	8	5	4	6	0.61	38	0.67	55	164	31
a145	*	6	9	1	7	9	0.60	39	0.71	29	138	13
a199		5	8	5	8	4	0.60	40	0.67	48	171	25
a1	*	7	8	5	7	3	0.60	41	0.67	56	156	28
a175		9	1	9	5	7	0.60	42	0.69	37	125	20
a174		9	1	9	5	7	0.60	43	0.69	39	125	20
a23		5	3	5	9	8	0.60	44	0.67	44	153	23
a237		5	9	6	4	6	0.60	45	0.67	54	155	20

Table 6-1. Decision matrix and results for a large problem

a45		8	7	3	5	7	0.60	46	0.67	52	146	24
a259		8	5	9	4	4	0.60	47	0.67	53	164	15
a16		9	6	7	3	5	0.60	48	0.67	50	149	17
a173		8	5	3	5	9	0.59	49	0.67	51	153	26
a3		6	5	7	4	7	0.59	50	0.64	61	149	30
a240		4	7	7	6	5	0.59	51	0.64	71	147	28
a63	*	9	8	9	2	3	0.59	52	0.69	42	146	14
a121		7	6	7	6	3	0.59	53	0.64	76	139	27
a56	*	2	8	9	3	9	0.59	54	0.69	41	151	16
a123	*	2	9	9	9	2	0.58	55	0.69	40	144	13
a256		5	3	9	5	7	0.58	56	0.64	63	144	26
a221		6	2	5	8	8	0.58	57	0.64	68	123	30
a262		7	1	8	7	6	0.58	58	0.64	70	105	35
a61		9	8	4	2	7	0.58	59	0.67	46	138	23
a11		8	5	8	6	2	0.58	60	0.64	73	139	27
a97		8	4	6	9	2	0.58	61	0.64	64	132	20
a34		7	4	9	1	9	0.58	62	0.67	45	121	15
a142	*	8	9	1	5	7	0.58	63	0.67	49	113	25
a58	*	7	3	3	7	9	0.58	64	0.64	59	127	32
a189		7	5	2	9	6	0.58	65	0.64	72	119	33
a157		9	7	5	6	2	0.57	66	0.64	75	126	30
a6	*	8	8	3	2	9	0.57	67	0.67	47	135	25
a96		9	3	3	6	8	0.57	68	0.64	69	138	24
a35		6	4	5	7	6	0.57	69	0.62	86	137	37
a90		7	5	5	6	5	0.57	70	0.62	93	141	34
a197		2	8	8	7	4	0.57	71	0.64	65	133	23
a141		6	5	5	5	7	0.57	72	0.62	91	138	41
a117		2	9	6	7	5	0.57	73	0.64	60	125	28
a254	*	6	1	4	9	9	0.57	74	0.64	57	113	31
a52		2	9	8	6	4	0.57	75	0.64	66	132	23
a218	*	6	7	9	1	6	0.57	76	0.64	74	115	33
a166		6	4	9	6	3	0.56	77	0.62	88	133	28
a24		7	4	8	1	9	0.56	78	0.64	67	114	28
a248		1	9	8	7	4	0.56	79	0.64	62	113	22
a230		7	7	2	8	4	0.56	80	0.62	81	112	38
a118		2	9	2	7	9	0.56	81	0.64	58	112	31
a36		7	6	5	2	8	0.55	82	0.62	90	119	49
a67	*	4	1	5	9	9	0.55	83	0.62	84	101	37
a109	*	3	1	8	7	9	0.55	84	0.62	83	104	34
a84		4	8	9	4	3	0.55	85	0.62	87	130	27
a114		8	7	6	1	6	0.55	86	0.62	89	103	47
a193		3	8	7	9	1	0.55	87	0.62	79	109	37
a64		5	8	7	1	7	0.55	88	0.62	92	101	48
a25		4	9	6	2	7	0.55	89	0.62	80	114	42
a249	*	8	8	3	8	1	0.55	90	0.62	78	101	36
a144	*	1	2	9	7	9	0.55	91	0.62	77	101	26
a235		6	9	1	8	4	0.55	92	0.62	82	106	37
a245		3	4	7	6	7	0.55	93	0.60	99	108	43

Table 6-1. (cont.) Decision matrix and results for a large problem

a92	*	9	9	1	7	2	0.54	94	0.62	85	95	30
a48		4	6	8	8	1	0.54	95	0.60	95	91	43
a191		4	7	8	7	1	0.54	96	0.60	101	88	52
a204		9	5	2	5	6	0.54	97	0.60	98	104	43
a27		4	4	9	2	8	0.53	98	0.60	94	111	40
a246		6	8	1	5	7	0.53	99	0.60	97	86	59
a184		9	7	3	2	6	0.53	100	0.60	100	100	46
a39		6	8	1	9	3	0.53	101	0.60	96	97	48
a225		3	8	9	2	5	0.53	102	0.60	102	101	34
a264		8	9	5	1	4	0.53	103	0.60	103	92	43
a261		8	2	8	5	3	0.52	104	0.58	111	84	49
a126		6	1	5	6	8	0.52	105	0.58	108	78	63
a133		7	5	7	6	1	0.52	106	0.58	113	83	62
a130		3	3	3	8	9	0.52	107	0.58	104	78	45
a183		2	7	7	5	5	0.52	108	0.58	110	88	57
a76		3	7	4	5	7	0.52	109	0.58	107	98	60
a82		5	2	9	1	9	0.52	110	0.58	105	83	46
a40		8	6	8	1	3	0.52	111	0.58	112	81	54
a28		5	8	3	3	7	0.51	112	0.58	109	94	59
a227		2	9	4	5	6	0.51	113	0.58	106	85	59
a116		6	2	6	9	2	0.51	114	0.56	114	81	56
a74		4	2	9	6	4	0.51	115	0.56	120	79	56
a165		7	1	4	7	6	0.51	116	0.56	121	66	76
a161		6	3	5	8	3	0.51	117	0.56	127	86	64
a258		8	5	5	6	1	0.50	118	0.56	130	73	69
a100		1	4	9	5	6	0.50	119	0.56	116	69	62
a250		1	7	7	9	1	0.50	120	0.56	123	60	67
a26		6	6	4	7	2	0.50	121	0.56	131	79	69
a156		9	4	1	4	7	0.50	122	0.56	124	70	64
a192		8	5	1	4	7	0.50	123	0.56	125	67	72
a205		2	8	8	3	4	0.50	124	0.56	115	81	55
a228		7	5	1	3	9	0.50	125	0.56	128	63	71
a224		3	8	4	6	4	0.50	126	0.56	118	84	68
a135		1	8	3	7	6	0.50	127	0.56	122	61	67
a206		2	7	4	5	7	0.50	128	0.56	126	74	73
a212		8	8	1	1	7	0.49	129	0.56	117	62	72
a255		6	8	3	1	7	0.49	130	0.56	129	68	78
a86		3	9	2	2	9	0.49	131	0.56	119	69	54
a112		2	3	8	9	2	0.49	132	0.53	132	63	60
a15		2	1	7	8	6	0.49	133	0.53	139	52	77
a242		4	1	1	9	9	0.49	134	0.53	133	51	65
a46		2	1	7	7	7	0.49	135	0.53	140	52	76
a29		3	1	6	5	9	0.49	136	0.53	134	61	79
a243		5	8	8	2	1	0.48	137	0.53	143	56	73
a169		5	3	5	6	5	0.48	138	0.53	144	72	78
a31		7	4	5	1	7	0.48	139	0.53	142	60	83
a77		8	7	5	2	2	0.48	140	0.53	136	62	77
a185		5	4	3	6	6	0.48	141	0.53	135	65	86

Table 6-1. (cont.) Decision matrix and results for a large problem

a38	4	4	6	3	7	0.48	142	0.53	137	67	82
a222	8	2	2	9	2	0.48	143	0.51	145	57	67
a127	6	7	4	4	3	0.47	144	0.53	141	63	83
a49	1	1	8	7	6	0.47	145	0.51	165	35	85
a195	9	1	6	5	2	0.47	146	0.51	154	44	84
a80	5	9	4	2	4	0.47	147	0.53	138	65	75
a188	6	2	9	4	2	0.47	148	0.51	158	58	72
a7	7	2	8	2	4	0.47	149	0.51	147	45	82
a73	7	2	8	2	4	0.47	150	0.51	167	45	82
a154	4	9	1	8	1	0.47	151	0.51	148	40	78
a140	5	1	3	6	8	0.47	152	0.51	163	47	92
a229	6	7	8	1	1	0.47	153	0.51	160	44	97
a266	3	4	2	9	5	0.47	154	0.51	162	55	71
a148	3	5	3	9	3	0.47	155	0.51	166	60	78
a128	7	4	6	5	1	0.46	156	0.51	151	47	85
a215	9	6	2	4	2	0.46	157	0.51	156	57	83
a155	7	2	6	4	4	0.46	158	0.51	153	47	92
a233	6	3	6	1	7	0.46	159	0.51	159	50	95
a10	4	7	4	7	1	0.46	160	0.51	161	53	93
a5	5	4	7	5	2	0.46	161	0.51	149	50	97
a119	5	3	6	6	3	0.46	162	0.51	155	49	96
a115	5	6	5	6	1	0.46	163	0.51	171	52	96
a71	4	5	8	3	3	0.46	164	0.51	150	58	78
a210	8	5	3	2	5	0.46	165	0.51	170	55	90
a72	2	7	7	2	5	0.46	166	0.51	164	51	87
a168	4	1	1	8	8	0.46	167	0.49	172	33	92
a87	2	6	7	3	5	0.46	168	0.51	168	49	86
a163	4	7	1	6	5	0.46	169	0.51	146	50	103
a178	5	7	3	5	3	0.45	170	0.51	169	57	96
a20	4	7	4	2	6	0.45	171	0.51	152	57	106
a62	3	7	5	4	4	0.45	172	0.51	157	54	102
a55	4	5	1	9	3	0.45	173	0.49	176	51	91
a47	9	5	5	1	2	0.45	174	0.49	182	46	95
a150	6	2	5	1	8	0.45	175	0.49	175	43	101
a149	1	1	8	8	3	0.45	176	0.47	184	32	103
a167	8	6	4	2	2	0.44	177	0.49	174	47	100
a93	2	1	2	8	8	0.44	178	0.47	183	32	95
a196	2	7	1	5	7	0.44	179	0.49	177	33	107
a152	7	4	5	4	2	0.44	180	0.49	178	47	107
a136	4	9	2	4	3	0.44	181	0.49	173	51	96
a103	1	1	6	5	8	0.44	182	0.47	188	30	113
a57	5	2	5	8	1	0.44	183	0.47	194	28	105
a132	5	6	6	3	2	0.43	184	0.49	181	42	114
a216	5	6	8	1	1	0.43	185	0.47	199	31	118
a14	4	2	1	7	7	0.43	186	0.47	185	26	116
a104	3	1	6	7	4	0.43	187	0.47	189	28	123
a43	1	9	6	3	2	0.43	188	0.47	187	33	100
a236	6	5	4	4	3	0.43	189	0.49	180	38	113

Table 6-1. (cont.) Decision matrix and results for a large problem

a202	4	4	5	3	6	0.43	190	0.49	179	42	119
a4	7	1	6	4	3	0.43	191	0.47	193	30	126
a81	1	3	9	6	1	0.43	192	0.44	205	27	105
a107	1	3	7	3	7	0.43	193	0.47	197	27	106
a113	1	3	6	7	4	0.43	194	0.47	191	27	108
a190	6	2	1	5	7	0.43	195	0.47	195	29	129
a162	6	4	2	1	8	0.43	196	0.47	186	35	121
a2	3	2	7	3	6	0.42	197	0.47	190	32	115
a105	9	4	5	1	1	0.42	198	0.44	210	31	122
a251	6	7	1	2	5	0.42	199	0.47	202	34	136
a68	4	2	5	3	7	0.42	200	0.47	198	29	120
a101	3	7	3	6	2	0.42	201	0.47	200	32	117
a32	6	7	4	2	2	0.42	202	0.47	196	32	128
a88	4	7	4	1	5	0.41	203	0.47	201	36	142
a125	6	5	7	1	1	0.41	204	0.44	208	23	136
a208	5	3	3	1	8	0.41	205	0.44	204	30	124
a94	4	6	7	1	2	0.40	206	0.44	203	26	138
a158	4	8	5	1	2	0.40	207	0.44	211	26	130
a78	6	1	6	3	4	0.40	208	0.44	212	23	148
a217	5	5	4	4	3	0.40	209	0.47	192	29	138
a194	7	6	2	1	4	0.40	210	0.44	209	27	138
a164	3	9	1	5	1	0.40	211	0.42	224	17	123
a232	4	1	1	5	8	0.40	212	0.42	214	18	145
a22	6	3	3	6	2	0.40	213	0.44	207	22	133
a171	2	6	8	1	2	0.40	214	0.42	220	20	132
a95	1	1	7	4	6	0.40	215	0.42	221	16	151
a151	9	2	1	5	1	0.40	216	0.40	234	15	140
a75	1	1	3	9	4	0.40	217	0.40	226	20	135
a223	1	3	8	2	5	0.40	218	0.42	218	23	131
a131	1	5	1	4	8	0.39	219	0.42	216	19	144
a160	4	9	3	2	1	0.39	220	0.42	217	17	127
a98	1	8	1	7	1	0.39	221	0.40	229	14	140
a159	2	3	5	1	8	0.39	222	0.42	215	23	135
a102	6	1	3	2	7	0.39	223	0.42	219	18	154
a108	5	3	3	3	6	0.39	224	0.44	213	25	139
a146	7	1	3	3	5	0.39	225	0.42	222	20	152
a51	6	1	3	7	1	0.38	226	0.40	237	19	154
a129	2	7	7	1	1	0.38	227	0.40	236	15	161
a153	2	1	3	8	4	0.38	228	0.40	230	23	144
a214	8	1	4	3	2	0.38	229	0.40	227	18	153
a33	8	1	2	4	3	0.38	230	0.40	228	18	149
a180	1	5	5	2	6	0.38	231	0.42	225	21	166
a198	4	4	4	4	4	0.38	232	0.44	206	18	167
a172	7	6	1	1	3	0.37	233	0.40	240	16	171
a99	3	8	1	1	5	0.37	234	0.40	238	16	160
a179	1	8	4	2	3	0.37	235	0.40	231	15	149
a211	5	5	2	2	5	0.36	236	0.42	223	21	169
a89	7	5	1	2	3	0.36	237	0.40	239	18	163

Table 6-1. (cont.) Decision matrix and results for a large problem

a120	1	3	7	3	4	0.36	238	0.40	233	17	159
a59	1	7	2	4	4	0.36	239	0.40	232	13	164
a252	1	9	1	1	4	0.36	240	0.36	247	8	157
a260	4	1	7	1	4	0.36	241	0.38	241	12	185
a234	4	7	2	3	2	0.35	242	0.40	235	16	166
a226	3	8	1	3	2	0.35	243	0.38	244	10	158
a220	2	1	1	9	1	0.35	244	0.31	256	7	149
a18	1	3	3	3	7	0.34	245	0.38	242	13	177
a238	1	7	2	3	4	0.34	246	0.38	243	9	176
a110	3	8	2	1	2	0.34	247	0.36	249	9	163
a231	2	4	3	1	6	0.31	248	0.36	246	7	204
a12	2	4	3	3	5	0.31	249	0.38	245	12	200
a147	6	1	1	5	1	0.31	250	0.31	259	4	230
a70	7	3	1	2	1	0.30	251	0.31	258	4	203
a124	2	2	5	4	3	0.30	252	0.36	248	6	213
a176	5	3	2	3	3	0.29	253	0.36	250	10	216
a21	1	3	1	1	7	0.29	254	0.29	264	2	209
a241	1	4	5	3	2	0.28	255	0.33	252	7	227
a201	4	1	2	4	4	0.28	256	0.33	251	6	228
a186	3	3	5	3	1	0.28	257	0.33	254	5	231
a137	6	3	2	1	2	0.28	258	0.31	257	4	214
a91	1	4	3	4	3	0.27	259	0.33	253	4	238
a17	2	1	2	5	4	0.27	260	0.31	255	3	231
a239	1	2	1	4	5	0.26	261	0.29	263	2	246
a177	1	5	2	1	4	0.25	262	0.29	261	2	245
a69	1	1	4	3	4	0.24	263	0.29	260	1	249
a207	1	5	3	2	1	0.23	264	0.27	266	1	251
a244	1	2	4	4	1	0.23	265	0.27	265	0	258
a134	3	3	2	2	3	0.21	266	0.29	262	0	250

Table 6-1. (cont.) Decision matrix and results for a large problem

6.5 Speed of execution

The execution time of several of the sample problem was bench-marked. The DOS system time was stored before and after each program was run. This was handled with the batch file shown in figure 6-2, to prevent unnecessary variations due to operator. The times indicate the total time needed to read a previously entered file from disk, execute the selected algorithm, save the results to disk, and terminate the program. The program "key-fake" was loaded prior to running each algorithm to automate keystrokes internal to the program, again to reduce needless variations in the data.

All timings were done on an IBM PC AT with standard hardware.

```
date2 > %1t.go
key-fake  "l"0 "f"0 "%1"13 "r"0 "d"0 "d"0 "%1t" 13 "d"0
"w"0 "a"0 "s"0 "q"0 "x"0 "y"0
topsis
date2 > %1t.stp
```

Figure 6-2. Batch file automating program bench marks

An attempt was made to bench-mark selectively larger problems to see if the actual results correlated with what the algorithms for each method would suggest. The results of timing each algorithm on each of five different datasets is shown in figure 6-3.

problem	alts	atts	DOMINANCE	ELECTRE	SAWM	TOPSIS
fighter	4	6	4.83	4.17	4.01	5.38
Crama	11	11	8.18	6.70	4.78	6.26
shoe	17	5	5.00	7.36	4.45	5.94
random	100	5	14.78	2:17.76	11.32	12.68
random	266	5	30.21	23:35.70	20.76	26.42
random	500	5	51.	1:22:02.	25.	36.
random	700	6	4:00.	3:10:13.	36.	56.
random	1000	5	1:32.	5:27:32.	44.	1:17.

Table 6-3 Benchmarks of MADM algorithms

The results agree fairly well with what was expected. Dominance and ELECTRE are of the order m^2 where m is the number of alternatives. The times for dominance are much lower for three reasons, dominance is $m^2/2$ where ELECTRE is strictly m^2. The calculations of concordance between two alternatives is more intensive than strictly dominance. And finally, once an alternative is dominated, it need not be compared again, where ELECTRE requires that all comparisons be carried out. The times for the dominance algortihm increase rapidly when more attributes are present (and therefore more nondominated alternatives). This is shown by test problem containing 700 alternatives and 6 attributes.

Because of computer memory constraints, the ELECTRE algorithm was re-written to solve the problems with over 100 alternatives. This caused an estimated two-fold increase in processing time for the problem. However, the times for ELECTRE are an order of magnitude slower than the other methods. This could be major deficit of this method; with a PC, though, it is really no problem to start the

problem in the evening and return to review the results the next day.

SAWM and TOPSIS are both of the order m. It is expected for large values of m that both be faster than dominance or ELECTRE. The table bears this out. TOPSIS takes longer than SAWM for two reasons. The decision matrix must be looked through once to find the positive and negative ideal solutions, and then a second time to calculate results, and secondly, TOPSIS requires more calculations with each alternative to reach its solution.

6.6 Space considerations of the algorithms

A criteria that is closely related to speed is space. As problem size increases, storage requirements and processing time increase proportionally. All the methods require m*n storage units for the decision matrix, and as the number of alternatives increases, this overshadows all the remaining variables. ELECTRE, with its concordance, discordance, and aggregate dominance matrices uses storage space at the rate of 3*m*m, this is much, much faster than any of the other three methods. This is a major deficit when attempting to use ELECTRE.

In order to save space when solving the problem containing 266 attributes necessitated rewriting the ELECTRE algorithm slightly and doubling the amount of time necessary to solve the problem.

6.7 Conclusions

When selecting a MADM method achieving the best solution is of primary concern. If our only desire is the top alternative, any of these four methods will give similar results; with the exception of ELECTRE and dominance. The solution set for these methods can be large, and there is no preference given between the alternatives in this set. The utility of ELECTRE drops quickly because of its inherent slowness. Dominance is not to be preferred if anything except the best alternative is desired (and it doesn't even give this result). SAWM is very good, but there is some concern over the comparability of attributes.

Our basic conclusion is this: TOPSIS is the method of choice for all MADM problems. TOPSIS is straightforward, fast, efficient, and not only does it select the same best methods as ELECTRE, SAWM, and DOMINANCE, it seems to be better at discriminating alternatives in the middle ranges, especially excellent alternatives not quite the top items, but close.

CHAPTER 7 Conclusions

The purpose of this study was to consider the practical applicability of MADM methods, their combined usefulness, and to enhance their ease of use.

7.1 Study review

In chapter 2 the eigenvector method is used to reduce the number of attributes of a DM problem prior to the collection or entry of the data into the computer in order to reduce the work necessary on large problems, and more importantly to allow more dominated alternatives to be discarded through the use of the dominance method. This seems to be a good concept, however, in the light of the results of chapter 6 (comparison of methods) we would not suggest the use of dominance as a method to reduce the number of alternatives to be considered. A better method would be the use of the simple additive weighting method or TOPSIS. With due consideration to the concept, it is our opinion that dominance is useless in actual applications.

The use of a multiple leveled attribute and weight generator similar to chapter 3 is logically appealing. It helps the DM group related attributes that are closer together. More research is needed in this area to further determine the practicalities of this type of MADM method.

ELECTRE is widely used in the European nations. It has a good theoretical base, the output is not as easily interpreted as SAWM or TOPSIS. In consideration of the storage requirements and speed of ELECTRE, we conclude that it is not the best method for large problems. The main difficulty with ELECTRE, however, is that the usual output of the set of non ELECTREcally dominated alternatives is not the most useful to the decision maker.

The simple additive weighting method (SAWM) has a good traditional backing, as it is one of the earliest developed. The results are readable, and easily interpreted. There is an absolute ranking given to each alternative. The only weak point in SAWM is its inattention to low attribute values.

TOPSIS overcomes this deficiency of SAWM, in that it does consider the negative ideal solution. TOPSIS is, in our opinion, the best MADM method in use today. It gives a specific preference order that has an absolute ranking.

7.2 Overall results

The combination of methods used in Chapters 2 and 3, will be useful with better selection of the methods used. The multiple level attribute and weight generation concept is powerful and useful. The hierarchal order formed with this method may be useful in generating weights for the use by SAWM or TOPSIS.

The best method for discriminating between alternatives is TOPSIS. TOPSIS has excellent theoretical backing, achieves results similar to SAWM, ELECTRE, and dominance for the top alternative, and gives a better score for discriminating between the alternatives directly below the top alternative. TOPSIS is also fast and doesn't use copious amounts of memory.

7.3 Considerations for future study

Although ELECTRE seems to be impractical on large problems because of its slowness, and because its output is not as easily understood as the output from TOPSIS or SAWM, the concept is quite useful and it seems to contain more information than SAWM or TOPSIS. The alternatives are all compared against each other, not just to the positive and negative ideal solutions as in TOPSIS. One would expect more information to be present, it mainly seems to be in the unusable form of belonging to the non-ELECTREcally dominated set or not belonging to this set. There are two possible avenues for making the existing information more usable. First, a ratio containing the number of the alternatives that dominate a specific alternative and the number of alternatives dominated by that alternative could be developed. Second, on small problems, an algorithm that dynamically changes the concordance and discordance thresholds by a small incremental amount and recalculated

the dominated alternatives could give the DM a better idea of the absolute placement of alternatives.

There may be room for some modification to the dominance method to enhance it's usefulness. Were strict dominance not adhered to, but some of the information on attribute weights used in the algorithm the results might become more usable. This is, however, what SAWM, ELECTRE, and TOPSIS attempt to do. Nevertheless, both dominance and ELECTRE promote the use of information that is not calculated in SAWM or TOPSIS: comparing each alternative to all others. Dominance has no concern for the weights. ELECTRE has no absolute priority. There is room for further consideration of how to use this information. and how to present it to the DM in an understandable and usable manner.

There is one final point that can be made, not directly related to the methods or their suitability. This concerns the data with which the methods work. With the proliferation of small computers, it is inevitable that more programs of this type will be developed. Some consideration should be made of a data standard for the transfer of information between these programs. The data stored by the programs here is compatible, but the author knows of two other existing MADM programs (for different methods) which do not have compatible files. The need for a standard is looming in the future of MADM programming.

REFERENCES

1. Calpine, H. C. and A. Golding, "Some Properties of Pareto-Optimal Choices in Decision Problems," OMEGA, Volume 4, Number 2, pp. 141-147.

2. Saaty, T. L. "A Scaling Method for Priorities in Hierarchical Structures," Journal of Mathematical Psychology, Vol. 15, No. 3, pp. 234-281, 1977

4. MacCrimmon, K. R., "Decision Making Among Mulit-Attributed Aternayives: A Survey and Consolidated Approach," RAND Memorandum, RM-4823-ARPA, 1968.

5. Churchman, C. W. and R. L. Ackoff, "An approximate Measure of Value," Journal of the Operations Research Society of America, Vol. 2, No. 2, pp. 172-187, 1954.

6. Klee, A. J., "The Role of Decision Models in the Evaluation of Competing Environmental Health Alternatives," Management Science, Vol. 18, No. 2, pp. B52-B67, 1971.

7. Yoon, K., "Systems Selection by Multiple Attribute Decision Making," Ph. D. Dissertation, Kansas State University, Manhattan, Kansas. 1980.

8. Hwang, C. L. and K. Yoon, "Multiple Attribute Decision Making: Methods and Applications. A State-of-the-Art Survey," Springer-Verlag, Berlin Heidelberg New York, pp. 128-140, 1981.

9. Apple, J. M. Plant Layout and Material Handling, Wiley, New York, 1977.

10. Fishburn, P. C., "Utility Models for Multiple Objective Decision: Do They Accurately Represent Human Preferences?," Decision Sciences, Vol. 10, No.3, pp. 451-479, 1979.

11. Miller, J. R. III, Professional Decision Making: A Procedure for Evalauating Complex Alternatives, Praeger, New York, 1970.

12. Crama, Y. and Hansen, P., "An Introduction to the ELECTRE Research Programme," Essays and Surveys on Multiple Criteria Decision Making, Springer-Verlag, Berlin Heidelberg New York, pp. 31-42, 1982.

13. Churchman, C. W., R. L. Ackoff, and E. L. Arnoff, Introduction to Operations Reaearch, Wiley, New York, 1957

14. MacCrimmon, K. R., "An Overview of Multiple Objective Decision Making" in Multiple Criteria Decision Making, University of South Carolina Press, Columbia, South Carolina, 1973, pp. 18-44.

Appendix A

Determination of Expected Number of Nondominated Alternatives

A spread sheet was developed to verify the expected number of nondominated alternatives when m alternatives are compared with respect to n attributes. The produce the results plotted in Chapter 2, Figure 1. The following approximation was used:

$$a(m,n) = 1 + \ln m + (\ln m)^2/2! + \ldots + (\ln m)^{n-3}/(n-3)! + q(\ln m)^{n-2}/(n-2)! + (\ln m)^{n-1}/(n-1)!$$

where q is Euler's constant (~0.5772), a is the average number of nondominated solutions, m is the number of initial alternatives, and n is the number of attributes.

The values 10, 100, and 1000 were used for m, the number of alternatives. The number of attributes, n, took the range from 2 to 10.

Spreadsheet Column	C	D	E	F	G	H	I	J	K
Row					Number of attributes				
2		2	3	4	5	6	7	8	9	10
5	1000	8	29	77	158	273	408	549	678	785
8	100	5	14	28	44	61	74	85	92	96
11	10	3	5	7	8	9	10	10	10	10

The values of the cells indicate the number of nondominated alternatives for several possible number of alternatives and attributes. For example, with 100 alternatives run through the dominance method with 4 attributes, it is expected that there will be approximately 28 nondominated alternatives (from cell E8).

The formulae in rows 4, 7, & 10 (results are not displayed) of the spread sheet are intermediate terms in the series. They are calculated separately to ease the summing of the terms for each equation. The formulae in rows 5, 8, & 11 are the series for each value of attribute, 2 to 10. The formulae in rows 4, 7, & 10 are identical, as are the formulae in 5, 8, & 11. The only difference is with the initial number of alternatives.

```
Cell Formulae:
  E1: 0.5772            Euler's constant
  C2-K2                 Number of attributes 2-10
  B4: (F0) 1000
  C4: (F1) @LN($B4)     Horton's method for series expansion
  D4: (F1) +C4*C4/2     is used to calculate the intermediate
  E4: (F1) +D4*C4/3     terms of the series. These terms are
  F4: (F1) +E4*C4/4     used as n, the number of alternatives,
  G4: (F1) +F4*C4/5     increases.
  H4: (F1) +G4*C4/6
  I4: (F1) +H4*C4/7
  J4: (F1) +I4*C4/8
  K4: (F1) +J4*C4/9
  B5: (F0) 1000
  C5: (F1) 1*$E$1+C4                Calculate each series.
  D5: (F1) 1+C4*$E$1+D4
  E5: (F1) 1+C4+D4*$E$1+E4
  F5: (F1) 1+C4+D4+E4*$E$1+F4
  G5: (F1) 1+C4+D4+E4+F4*$E$1+G4
  H5: (F1) 1+C4+D4+E4+F4+G4*$E$1+H4
  I5: (F1) 1+C4+D4+E4+F4+G4+H4*$E$1+I4
  J5: (F1) 1+C4+D4+E4+F4+G4+H4+I4*$E$1+J4
  K5: (F1) 1+C4+D4+E4+F4+G4+H4+I4+J4*$E$1+K4
```

Appendix B

Quantification of Subjective Attribute Values.

One of the common way of converting a qualitative attribute into a numerical value is using a 10-point Bipolar scale [4] [9]. The endopints would be given the values of 10 and 0, representing the maximum and mimimum attribute values that are physically realizable. The midpoint would be the basis of calibrating. Values above it would be more favorable and values below would be less favorable. This type of scaling works well with most applications and is used with many MADM methodoligies.

There are ocassionally attributes that would be poorly quantified with a Bipolar scale and would be better served with other scales [Miller, 11]; but for our purposes here we will use the Bipolar scale.

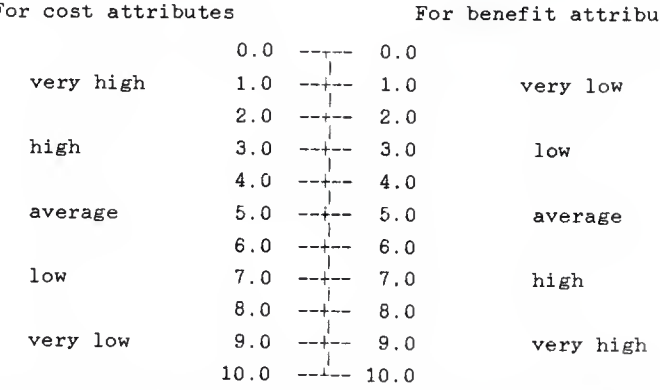


Figure B-1. Assignment of values for an interval scale.

Appendix C

Procedures for specific algorithms

C-1. The PASCAL procedure implementing the DOMINANCE method:

It is initially assumed that there are no dominated alternatives in the set. Dominance is checked by comparing the first alternative with each successive alternative of the set, discarding dominated alternatives. Then, the second alternative is compared against the remaining nondominated alternatives; again, discarding dominated alternatives. And so on to the end of the list. Finally, the next-to-last nondominated alternative is compared to the last nondominated alternative to give the entire nondominated set; all dominated alternatives have been discarded. There are a maximum of $n(n-1)/2$ comparisons in a list of n alternatives.

The specific algorithm proceeds as follows: The variable "isDominated" is an array of boolean elements with each element representing an alternative to be tested for dominance. Initially, every element in the array is set to false, i.e. no alternatives are dominated. Each member of the set of alternatives is then checked against successive members of the set for dominance. Dominated alternatives are flagged in the isDominated array. The variable altnum is an integer variable containing the number of alternatives in the set.

```
procedure dominance;
begin  { dominance }
```

```
{ no alternatives are yet dominated }
  for k := 1 to altnum do isDominated[k] := false;
{ check for dominance }
  for j := 1 to altnum-1 do
  begin
    k := j;
{
if the checking alternative is dominated (j) stop using
it for comparison purposed
}
    while (not isDominated[j]) and (k < altnum) do
    begin
      k := k + 1;
{
if alternative k isn't already dominated
}
      if not isDominated[k]
      then begin
{
Check to determine if alternative j or k is dominated.
If one of alternatives j and k are dominated, change
the isDominated list to indicate the dominated
condition
}
        if dominated(j,k,which)
        then isDominated[which] := true;
      end; { if not isDominated[k] }
    end; { while }
  end; { for j := 1 to altnum-1 }
end;   { procedure dominance }
```

Listing 2-1. The dominance procedure.

The function "dominated" is used to determine if one of the two alternatives being checked, a and b, is dominated by the other, and if so, which one is the dominated alternative. The function returns the value of true if an alternative is dominated, and returns the number of the dominated alternative in the pass by reference parameter "which". The number of attributes in the is stored in the variable "altnum", the flag indicating a cost or benefit

criterion for each attribute is stored in the array "cost benefit". The data delineating the attribute values for each alternative are stored in the matrix "decmat"

```
function dominated(    a, b: integer;
                       var which: integer): boolean;
var
  g,h : integer;
  ab, ba : boolean;
begin
  which := 0;
  ab := true;
  ba := true;
  for h:=1 to atbnum do
  begin
      ab := ab and (cost_benefit[h]*decmat^[a,h] >=
                    cost_benefit[h]*decmat^[b,h]);
      ba := ba and (cost_benefit[h]*decmat^[a,h] <=
                    cost_benefit[h]*decmat^[b,h]);
  end; { for h:= 1 to atbnum }
  dominated := ab xor ba;
{
Use exclusive or in case a and b have identical
attribute values
}
  if ab xor ba
  then if ab
       then which := b
       else which := a;
end; { function dominated }
```

Listing 2-2. The function Dominated.

C-2. The PASCAL procedure implementing the SAWM algorithm:

The SAWM algorithm is relatively easy to code in PASCAL. The attribute weights are checked to prevent division by zero, the decision matrix is normalized and then multiplied by the attribute weights to produce the weighted normalized decision matrix. These weights are then summed across alternatives, giving the SAWM weighted average for each alternative.

```
procedure sawm;
var
  g,h : integer;
  wnormat : pointamat; {weighted normalized decision
                        matrix}
  sumwght : real;
begin
  if not entered_DM
  then WaitMsg('Data not entered')
  else begin
    new(wnormat);
    Msg('Please Wait, Simple Additive Weighting Method
         executing.');
    sumwght:=0;
    for h:=1 to atbnum do sumwght:=sumwght+dmwght[h];
    if sumwght = 0
    then for h:= 1 to atbnum do
      dmwght[h]:=1/atbnum;
    wnormat^:=decmat^;
    normal(wnormat^,altnum,atbnum);
    for g:=1 to atbnum do value^[g]:=0;
    for g:=1 to altnum do for h:=1 to atbnum do
      value^[g]:=value^[g]+wnormat^[g,h]*dmwght[h];
    dispose(wnormat);
  end; {if entered}
end;   {sawm}
```

This procedure normalizes the decision matrix, using the proper linear scale transformations for each attribute, whether benefit or cost criteria.

```
procedure normal(var inp:amat;i,j:integer);
var
  g,h : integer;
  nMax : atbvect;
begin
  for h:=1 to j do nMax[h]:=inp[1,h];
  for g:=1 to i do
    for h:=1 to j do
      if   cost_benefit[h] = 1
      then begin
         if   nMax[h] <  inp[g,h]
         then nMax[h] := inp[g,h]
      end
      else begin
         if   nMax[h] >  inp[g,h]
         then nMax[h] := inp[g,h]
      end;
```

```
    for g:=1 to i do
      for h:=1 to j do
      if nMax[h] <> 0
      then begin
        if   cost_benefit[h] = 1
        then inp[g,h]:=inp[g,h]/nMax[h]
        else inp[g,h]:=nMax[h]/inp[g,h];
      end;
  end;
```

C-3. The PASCAL procedures implementing the multiple level attribute and weight generation method.

The structure of the data in the hierarchy is fundamentally different from the data structures used by the other methods. The local weights are calculated for each level from the pairwise comparison of the levels' attributes.

```
procedure figureEigenWeights;
var
  g, h : integer;
begin
  with GlData do
  begin
    if entered
    then CalcEigen(Eig, Weight, NumItem, Consty)
    else for h := 1 to numItem do
           weight[h] := 1/numItem;
    for g := 1 to numItem do
      if (GlData.Leaf[g].L_numItem <> 0)
      then with GlData.Leaf[g] do
        if entered
        then CalcEigen(L_Eig, L_weight,
                       L_numItem, L_Consty)
        else for h := 1 to L_numItem do
          L_weight[h] := 1/L_numItem;
  end; { with GlData do }
end;

procedure TotalWeights;
var
  g, h : integer;
begin
  with GlData do
```

```
    begin
      for g := 1 to numItem do
        if (GlData.Leaf[g].L_numItem <> 0)
        then with GlData.Leaf[g] do
          for h := 1 to L_numItem do
            L_lWeight[h] := L_weight[h] *
                            GlData.Weight[g];
    end; { with GlData do }
  end;
    procedure hawmII;
    begin
      if EnteredEigenVectors and EnteredAttributeNames
      then begin
        Msg('Executing');
        figureEigenWeights;
        TotalWeights;
        Waitmsg('Done');
      end
      else WaitMsg('Data not entered');
    end;
```

C.5 The PASCAL procedure implementing the ELECTRE algorithm:

This procedure implements the ELECTRE method. The method is discussed in "Multiple Attribute Decision Making - Methods and Applications," by C. L. Hwang and K. Yoon, Springer-Verlag, Berlin, 1981, pp 115f. This implementation of the ELECTRE algorithm is optimized to save time and storage. The normalized decision matrix and the concordance/discordance dominance matrices are not stored. The concordance and discordance sets are also not stored. This is facilitated by calculating each pairwise concordance and discordance value and immediately storing it in the respective concordance or discordance matrix; which requires no intermediate storage of the concordance/ discordance sets for each pair of alternatives. When calculating the aggregate matrix, the concordance/discordance matrices are not evaluated as a whole, to determine the concordance/ discordance dominance matrices, but each individual dominance value (0 or 1) is evaluated and used to determine the value of the corresponding item in the aggregate dominance matrix.

```
procedure electre;
{
The function max determines which of two real values has
the greatest absolute value, and assigns that absolute
value to the real variable max.
}

function max(a, b: real): real;
begin
```

```
  if abs(a) >= abs(b)
  then max := abs(a)
  else max := abs(b);
end;
{
Normalizes the decision matrix according to the formula:

          a[i,j] =  a_ij / √ [ Σ_{i=1}^{m} (a_ij)^2 ]

}
procedure normal(var inp:amat;i,j:integer);
var
  g,h : integer;
  normalSum : atbvect;
begin
  for h:=1 to j do normalSum[h]:=0;
  for g:=1 to i do for h:=1 to j do
    normalSum[h]:=normalSum[h]+sqr(inp[g,h]);
  for h:=1 to j do normalSum[h]:=sqrt(normalSum[h]);
  for g:=1 to i do
    for h:=1 to j do
      if normalSum[h] <> 0
      then inp[g,h]:=inp[g,h]/normalSum[h];
end;

var
  g,h,j,k,l : integer;
  sumwght : real;
  concordVal,              {temporary holder for 0, 1 values}
  discordVal: integer;     { for aggregate dominance matrix }
  meanConcord,             {mean concordance value of matrix}
  meanDiscord,             {mean discordance value of matrix}
  totalConcord,            {total discordance value of matrix}
  totalDiscord,            {total discordance value of matrix}
  maxN,             {numerator for calculating discordances}
  maxD : real;    {denominator for calculating discordances}

begin
  if not entered_DM
  then WaitMsg('Data not entered')
  else begin
    Msg('Please Wait . . .Electre executing. . . .  ');
{
Ascertain that the attribute weights do not total zero.  If
they do, assign equal non-zero values that sum to one.
}
    sumwght:=0;
    for h:=1 to atbnum do sumwght:=sumwght+dmwght[h];
    if sumwght = 0
    then for h:= 1 to atbnum do
           dmwght[h]:=1/atbnum;
{
```

```
Copy the decision matrix, normalize and weight the copy.
}
    wnormat^:=decmat^;
    normal(wnormat^,altnum,atbnum);
    for g:=1 to altnum do
      for h:=1 to atbnum do
        wnormat^[g,h]:=wnormat^[g,h]*dmwght[h];
{
Calculate the concordance and discordance matrices on the
fly; don't save the values from each pairwise comparison of
alternatives, but immediately calculate and store the
values in the proper matrix.  Concurrently, calculate the
total concordance and discordance in order to speed
calculation of the mean.
}
    totalConcord := 0;
    totalDiscord := 0;
    for k := 1 to altnum do for l := 1 to altnum do
    begin
      concord^[k,l] := 0;
      discord^[k,l] := 0;
      if k <> l
      then begin
        maxN   := 0;
        maxD   := 0;
        for j := 1 to atbnum do
        begin
          if cost_benefit[j]*wnormat^[k,j] >=
             cost_benefit[j]*wnormat^[l,j]
          then concord^[k,l] := concord^[k,l] + dmwght[j]
          else maxN :=
                   max(maxN, wnormat^[k,j]-wnormat^[l,j]);
          maxD := max(maxD, wnormat^[k,j]-wnormat^[l,j]);
        end;
        discord^[k,l] := maxN/maxD;
        totalConcord := totalConcord + concord^[k,l];
        totalDiscord := totalDiscord + discord^[k,l];
      end; { if j <> l }
    end; { for l := 1 to altnum }
{
Calculate the average concordance/discordance index.
}
    meanConcord := totalConcord/(altnum*(altnum-1));
    meanDiscord := totalDiscord/(altnum*(altnum-1));
{
Calculate the aggregate dominance matrix, keeping the
concordance and discordance matrices undominated by storing
the dominance values in the temporary variables concordVal
and discordVal.  This will allow for later changes to the
program to vary the threshold values from the average
concordance index.
}
```

```
    for k := 1 to altnum do for l := 1 to altnum do
      if k = l
      then aggregate^[k,l] := 0
      else begin
        if concord^[k,l] < meanConcord
        then concordVal := 0
        else concordVal := 1;
        if discord^[k,l] > meanDiscord
        then discordVal := 0
        else discordVal := 1;
        aggregate^[k,l] := concordVal*discordVal;
      end { else begin }
  end; {if entered}
end;   {electre}
```

C.6 The PASCAL procedure implementing the TOPSIS algorithm:

```
procedure topsis;

var

  g,h : integer;

  wnormat : pointamat;{weighted normalized D matrix}

  sumwght : real;

begin { topsis procedure }

  if not entered_DM

  then WaitMsg('Data not entered')

  else begin

    new(wnormat);

    Msg('Please Wait . . .Topsis executing. . . .  ');

    sumwght:=0;

    for h:=1 to atbnum do sumwght:=sumwght+dmwght[h];

    if sumwght = 0

    then for h:= 1 to atbnum do dmwght[h]:=1/atbnum;

    wnormat^:=decmat^;

    normal(wnormat^,altnum,atbnum);
```

```
    for g:=1 to altnum do
      for h:=1 to atbnum do
        wnormat^[g,h]:=wnormat^[g,h]*dmwght[h];
    ideal(wnormat^,altnum,atbnum,pisoln,nisoln);
    for g:=1 to altnum do spid^[g]:=0;
    for g:=1 to altnum do snid^[g]:=0;
    for g:=1 to altnum do for h:=1 to atbnum do begin
      spid^[g]:=spid^[g]+sqr(wnormat^[g,h]-pisoln[h]);
      snid^[g]:=snid^[g]+sqr(wnormat^[g,h]-nisoln[h]);
    end;
    for g:=1 to altnum do begin
      snid^[g]:=sqrt(snid^[g]);
      spid^[g]:=sqrt(spid^[g]);
      end;
    for g:=1 to altnum do
      value^[g]:=snid^[g]/(spid^[g]+snid^[g]);;
    {leave positive ideal and negative non normalized}
    ideal(decmat^,altnum,atbnum,pisoln,nisoln);
    dispose(wnormat);
  end; {if entered}
end;   {topsis}
```

The following two procedures are called by the TOPSIS procedure. The first one is called to normalize the decision matrix, and the second to determine the positive and negative ideal solutions.

```
{Normalizes the Decision Matrix}
procedure normal(var inp:amat;i,j:integer);
var
  g,h : integer;
  nsum : atbvect;
begin
  for h:=1 to j do nsum[h]:=0;
  for g:=1 to i do for h:=1 to j do
nsum[h]:=nsum[h]+sqr(inp[g,h]);
  for h:=1 to j do nsum[h]:=sqrt(nsum[h]);
  for g:=1 to i do for h:=1 to j do if nsum[h] <> 0
then inp[g,h]:=inp[g,h]/nsum[h];
end;

{calculates ideal and negative-ideal solution}

procedure ideal(var inp:amat;
                    isize,jsize:integer;
                var max,min:atbvect);
var
  g,h : integer;
begin
  for g:=1 to jsize do begin
    max[g]:=inp[1,g];
    min[g]:=inp[1,g];
```

```
  end;
  for g:=1 to jsize do
    for h:=1 to isize do begin
      if (inp[h,g]*cost_benefit[g]) >
         (max[g]*cost_benefit[g])
      then max[g]:=inp[h,g];
      if (inp[h,g]*cost_benefit[g]) <
         (min[g]*cost_benefit[g])
      then min[g]:=inp[h,g];
  end;
end;
```

```
File: dom2.pas
{$R+,C-,V-}
{
  This program executes the DOMINANCE algorithm.  This program is menu
driven.  The arrow keys on the numeric keypad will move the blinking cursor
to  the option you desire.  The return key, <ret>, then selects that
option.  Department of Industrial Engineering Kansas State University.
  The program was written by Dan McAnarney.
}
{ Copyright 1986  Version 2.04   }

program dominant(input,output);
const
  altmax     = 1000;         {Maximum number of alternatives }
  atbmax     =   10;         {Maximum number of attributes   }
  G_scwidth  =   80;         {Screen width (or printer width }
  naltlen    =   30;         {fieldwidth for strings }
  natblen    =    9;         {fieldwidth for strings }
  lines      =   15;
  blank15    =   '';
type
  saveType   = (all,some);
  charset    = set of char;
  pointamat  = ^amat;
  amat       = array [1..altmax,1..atbmax]of real;
  pointBmat  = ^amat;
  bmat       = array [1..atbmax,1..atbmax]of real;
  pointAlt   = ^altvect;
  altvect    = array [1..altmax] of real;
  atbvect    = array [1..atbmax] of real;
  atbvectboolean   = array [1..atbmax] of boolean;
  ivect      = array [1..atbmax] of integer;

  namealt    = string[naltlen];
  nameatb    = string[natblen];

  AnyString = string[70];
  String3   = string[3];
  saltvect  = array [1..altmax] of namealt;
  satbvect  = array [1..atbmax] of nameatb;

var
  decmat    : pointamat;           {decision matrix}
  nonDomAltnum : integer;      {number of non-dominated alternatives}
  altnum    : integer;         {number of alternatives}
  atbnum    : integer;         {number of attributes}
  isdominated : array[1..altmax] of boolean;
  altn      : saltvect;        {labels for alternative names}
  attb      : satbvect;        {labels for attribute names }
  cost_benefit : ivect;      { 1-benefit, -1-cost}
```

```
  FileName  : string[70];
  Line      : string[100];
  domfile   : text;           {output disk file. save data }
  Odev      : text;           {Output device, 'lst:','scn:'}
  consty    : real;           {consistency of the eigenvector}
  G_owidth  : integer;        { output device width}
  G_CloseOdev     : boolean;
  entered_DM      : boolean;
  G_InsertModeOn : boolean;
  enteredAlternativeNames : boolean;
  enteredAttributeNames : boolean;

{$I \common\numeric.inc}
{$I \common\getfname.inc}
{$I \common\menu.inc}
{$I \common\config.inc}
{$I \common\inpstrng.inc}
{$I \common\scrnprtr.inc}

{$I domfile.inc}

{$I domechos.inc}

{when integer input variables are are known to be between two
predetermined values, this procedure is called.  For example, the number of
alternatives must be between 2 and altmax (currently 200), so limitin is
called with altnum as the variable returned, 200 and 2 as the limits}

procedure limitin(var inval:integer;max,min:integer);
var
  ok : boolean;
begin
  repeat
    read_int(inval,ok);
    if not ok or (inval>max) or (inval<min)
    then begin
      Msg2('Maximum and Minimum values are: ');
      write(max:5,min:5,'  ');
    end;
  until (inval>=min) and (inval<=max);
end;

{$I namein.inc}
{$I domprint.inc }
{$I mdmedit.inc}
{$I dominput.inc}
{$I domalg.inc}
{$I common\exit.inc}
{$I common\logodom.inc}
{$I common\protect.inc}

{initialize variables for startup of new problem}
```

```
procedure init;
var
  g,h : integer;
begin
  for g:= 1 to altmax do for h:= 1 to atbmax do decmat^[g,h]:=0;
  for g:= 1 to altmax do altn[g]:=blank15;
  for g:= 1 to altmax do isDominated[g]:=false;
  for g:= 1 to atbmax do attb[g]:=blank15;
  for g:= 1 to atbmax do cost_benefit[g]:=1;
  altnum := 0;
  atbnum := 0;
  entered_DM     := false;
  enteredAlternativeNames := false;
  enteredAttributeNames := false;
end;

procedure start;
begin
  init;
  Msg(' Keyboard or File input? (K/F)');
  repeat Read(Kbd,Ch) until Upcase(Ch) in ['K','F'];
  Write(Upcase(Ch));
  if UpCase(Ch)='F'
  then load
  else kbdin;
end;

procedure main;
var
  menu_item : integer;
  all_done  : boolean;
  good_eqns : boolean;

procedure nomemory;
begin
  Msg('Not enough memory available to run program.');
  Halt;
end;

begin (* main *)
  {$I-}
  new(decmat);
  if IOResult<>0 then nomemory;
  {$I+}
{
  Protect;
}
  logodom;
  LoadConfig;
  G_InsertModeOn := true;
  G_CloseOdev    := true;
  all_done       := false;
```

```
  menu_item      := 1;
  init;
  screen_out(G_CloseDdev);
  repeat
    clrscr;
    textcolor(G1.MainMenuHeadingColor);
    gotoxy(1,1);
    Writeln('   Main Menu');
    Writeln;
    textcolor(G1.MainMenuItemColor);
    writeln(' Load new dataset );
    writeln(' Run DOMINANCE');
    writeln(' Display results');
    writeln(' Edit input data');
    writeln(' Save current dataset');
    writeln(' Configure system');
    writeln(' eXit program');
    lookmenu(7,menu_item,0,2,'LRDESCX');
    case menu_item of
      1 :Start;
      2 :dominance;
      3 :Display;
      4 :Kbdin;
      5 :SaveMenu;
      6 :Config;
      7 :exxit(all_done);
    end;
  until all_done;
end;

begin
  main;
end.

File: domechos.inc
{  matrix echo -used to print the matrix on the printer and in the
display mode of TOPSIS. This proc is not used in the EDIT mode.  }

procedure me2(var a: amat; all: boolean);
var
 g,h,i,j : integer;
 k,l     : integer;
 pfields : integer;
begin
  pfields:=((G_owidth-naltlen) div G1.fwidth)-1;
  if pfields <= 0 then pfields := 1;
  h:= atbnum div pfields;
  if (atbnum MOD pfields) <> 0 then h:=h+1;
  for g:=1 to h do begin
    k:= pfields*(g-1)+1;
    l:= pfields*g;
    if l > atbnum then l:=atbnum;
```

```
    write(Odev,'':naltlen,'   ');
    for i:= k to 1 do write(Odev,Copy(Concat(attb[i],'
'),1,Gl.fwidth),' ');
    writeln(Odev);
    for i:= 1 to altnum do
      if all or (not all and not isDominated[i])
      then begin
        write(Odev,altn[i]:naltlen);
        write(Odev,'  ');
        for j:= k to 1 do write(Odev,a[i,j]:Gl.fwidth:Gl.decdis,' ');
        writeln(Odev);
      end;
    writeln(Odev);
  end;
end;

File: domfile.inc
{
Saves dataset out to file.  If file already exists, prompts with a
"Replace Yes/No" message.
}

procedure Save(which: saveType);
var
  g,h,i: integer;
  ItemLength : integer;
  ok  : boolean;
  SavedAlts : integer;
begin
  repeat
    Msg('Save: Enter filename  ');
    GetFileName(Filename,'mdm');
    ok := true;
    if exist(FileName)
    then begin
      Msg2('File already exists. Replace (Y/N) ');
      repeat read(Kbd,ch); ch := Upcase(ch); until ch in ['Y','N'];
      if ch ='Y' then ok := true else ok := false;
      end;
  until ok;
if FileName<>'' then
  begin
    Assign(domfile,FileName);
    Rewrite(domfile);
    if which = all
    then SavedAlts := altnum
    else SavedAlts := nonDomAltnum;
    writeln(domfile,SavedAlts:15,atbnum:15);
    for g:=1 to altnum do
    begin
      if (which = all) or ( (which = some) and not isDominated[g] )
      then begin
```

```
        for h:=1 to atbnum do
        writeln(domfile,decmat^[g,h]:15:10);
      end;
    end;
    for g:=1 to atbnum do for h:=1 to atbnum do writeln(domfile,'    0.00');
    for h:=1 to atbnum do
    begin
      ItemLength := Length(attb[h]);
      writeln(domfile,attb[h]:ItemLength);
    end;
    for h:=1 to atbnum do
      writeln(domfile,'    0.00');
    for h:=1 to atbnum do
      writeln(domfile,cost_benefit[h]:5);
    for g:=1 to altnum do
    begin
      if (which = all) or ( (which = some) and not isDominated[g] )
      then begin
        ItemLength := Length(altn[g]);
        writeln(domfile,altn[g]:ItemLength);
      end;
    end;

    g := 0; h := 0; i := 0;
    if not enteredAlternativeNames then g := -1;
    if not enteredAttributeNames then h := -1;
    if not entered_DM then i := -1;
    writeln(domfile,g:5,h:5,i:5);

    Close(domfile);
  end;
end;

procedure SaveMenu;
var
  ix : integer;
  done : boolean;
begin
  ix:=1;
  done:=false;
  repeat
    clrscr;
    gotoxy(1,1);
    textcolor(Gl.SubMenuHeadingColor);
    Writeln(' Save Menu');
    Writeln;
    textcolor(Gl.SubMenuItemColor);
    writeln(' save all Alternatives');
    writeln(' save Non-dominated alternatives');
    writeln(' Quit - return to main menu');
    lookmenu(3,ix,0,2,'ANQ');
    case ix of
```

```
      1 : Save(all);
      2 : Save(some);
      3 : done := true;
    end;
  until done;
end;


{
Loads dataset from a file.  Prompts for a filename.  File has to exists
to be read.
}

procedure Load;
var g,h,i : integer;
begin
  listfiles('*.mdm');
  Msg('Load: Enter filename  ');
  GetFileName(Filename,'mdm');
  if (Filename<>'') then if (not exist(FileName)) then
  repeat
    Msg('File not Found: Enter another filename  ');
    GetFileName(Filename,'dom');
  until exist(FileName) or (FileName='');
  if FileName<>'' then
  begin
    ClrScr;
    Msg('Please Wait. Reading File...');
    Assign(domfile,FileName);
    Reset(domfile);
    read(domfile,altnum);
    nonDomAltnum := altnum;
    read(domfile,atbnum);
    for g:=1 to altnum do
      for h:=1 to atbnum do read(domfile,decmat^[g,h]);
    readln(domfile);
    for g:=1 to atbnum do for h:=1 to atbnum do readln(domfile);
    for g:=1 to atbnum do
      readln(domfile,attb[g]);
    for h:=1 to atbnum do
      readln(domfile);
    for g:=1 to atbnum do
      readln(domfile,cost_benefit[g]);
    for g:=1 to altnum do
      readln(domfile,altn[g]);

    readln(domfile,g,h,i);

    enteredAlternativeNames := (g = 0);
    enteredAttributeNames := (h = 0);
    entered_DM := (i = 0);
```

```
    Close(domfile);
  end;
end; {of Procedure Load}

File: dominput.inc

const
  lr1 = 16;

procedure MakeForm;
var
  g : integer;
begin
  clrscr;
  gotoxy(1,1);
  Writeln('Alternative # ');
  Writeln;
  Write('Name     :');
  clreol;
  Writeln;
  Writeln('Attributes: ');
  for g := 1 to atbnum do
  begin
    Write(attb[g]:natblen,' :');
    clreol;
    Writeln;
  end;
end;

procedure clearForm(num: integer;displayname: boolean);
var
  g : integer;
begin
  gotoxy(15,1);write(num);
  gotoxy(15,3);clreol;if displayname then Write(Altn[num]);
  for g := 1 to atbnum do
  begin
    GotoXY(lr1,g+4);clreol;
  end;
end;

procedure DisplayForm(num : integer);
var
  g : integer;
begin
  for g := 1 to atbnum do
  begin
    GotoXY(lr1-6,g+4);
    Write('[',decmat^[num,g]:61.fwidth:61.decdis,']');
  end;
end;
```

```
procedure readForm(num: integer; readname: boolean);
var
  g : integer;
  temp : real;
  ok, done : boolean;
begin
  done := false;
  if readname
  then begin
    gotoxy(12,3);clreol;
    InputStr(altn[num],naltlen,12,3,[^M],ch,G_InsertModeOn);
    if (altn[num] = '') then done := true;
  end;
  if not done
  then for g := 1 to atbnum do
       begin
         GotoXY(Ir1,g+4);
         read_real(temp,ok);
         if ok then decmat^[num,g]:=  temp;
       end;
end;
procedure matin(var List: amat);
var
  g : integer;
begin
    textcolor(Gl.EditHeadingColor);
    gotoxy(1,1);
    clrscr;
    MakeForm;
    textcolor(Gl.EditItemColor);
    for g := 1 to altnum do
    begin
      ClearForm(g,true);
      ReadForm(g,false);
    end;
    entered_DM := true;
end;

procedure matin2(var List: amat; var curr_num: integer; var name:saltvect);
var
  g : integer;
begin
    textcolor(Gl.EditHeadingColor);
    gotoxy(1,1);
    clrscr;
    MakeForm;
    textcolor(Gl.EditItemColor);
    g := altnum;
    repeat
      g := g + 1;
      ClearForm(g,false);
      ReadForm(g,true);
```

```
    until altn[g] = '';
    altnum := g - 1;
    entered_DM := altnum <> 0;
end;

{This procedure handels all keyboard data input.  Data can be keyed in
 to begin a new problem; or data in an existing problem can be edited.}
procedure kbdin;
var
  ix   : integer;
  done : boolean;
  g, h : integer;


{Maximize or Minimize each attribute variable}
procedure maxminin;
var
  tch : char;
begin
repeat
  clrscr;
  for h:=1 to atbnum do begin                                    {Max or minimize}
    Write('Maximize(Y/N) Attribute #');
    write(h:2,'  ',attb[h]);
    if cost_benefit[h]=1 then tch:='Y' else tch:='N';
    write(' [',tch,']');
    repeat Read(Kbd,Ch) until Upcase(Ch) in ['Y','N',^M];
    Writeln(Upcase(Ch));
    if upcase(ch)='Y' then cost_benefit[h]:=1;
    if upcase(ch)='N' then cost_benefit[h]:=-1;
  end;
  Msg('Are these values correct? (Y/N)');
  repeat Read(Kbd,Ch) until Upcase(Ch) in ['Y','N'];
  Writeln(Upcase(Ch));
until Upcase(Ch) ='Y';
end;

procedure MethodMatin;
begin
  if entered_DM
  then edit1(decmat^)
  else if enteredAttributeNames
       then if enteredAlternativeNames
            then begin
              matin(decmat^);
              entered_DM:=true;
            end
            else begin
              matin2(decmat^,altnum,altn);
              entered_DM:=true;
            end
       else WaitMsg('Alternative and Attribute Names Must be entered
first.');
```

```
end;


{Make sure decisiom matrix has been entered.}
procedure return;
begin
  if entered_DM
  then done:=true
  else begin
    Msg('You MUST enter the Decision Matrix. Press any key to continue. . .
');
    Read(Kbd,Ch);
  end;
end;

begin { kbdin }
  ix:=1;
  done:=false;
  repeat
    clrscr;
    gotoxy(1,1);
    textcolor(Gl.SubMenuHeadingColor);
    Writeln(' Edit/Input Data');
    Writeln;
    textcolor(Gl.SubMenuItemColor);
    writeln(' aLternative names edit or delete');
    writeln(' aTtribute names edit or delete');
    writeln(' Cost-benefit');
    writeln(' Decision matrix');
    writeln(' add alterNative names to existing dataset');
    writeln(' add attriBute names to existing dataset');
    writeln(' Quit - return to main menu');
    lookmenu(7,ix,0,2,'LTCDNBQ');
    textcolor(Gl.EditItemColor);
    case ix of
      1 : if EnteredAlternativeNames
          then EditAlternativeNames(altn,altnum,altmax,naltlen)
          else EnterAlternativeNames(altn,altnum,altmax,naltlen);
      2 : if EnteredAttributeNames
          then EditAttributeNames(attb,atbnum,atbmax,natblen)
          else EnterAttributeNames(attb,atbnum,atbmax,natblen);
      3 :maxminin;
      4 :MethodMatin;
      5 :matin2(decmat^,altnum,altn);
      6 :EnterAttributeNames(attb,atbnum,atbmax,natblen);
      7 :done := true;
    end;
  until done;
end;

File: domprint.inc
procedure Display;
var
```

```
  g,h : integer;
  n   : pointamat;              {normalized decision matrix}
  alldone    : boolean;
  prin_num   : integer;


begin
  OutputDevice(G_CloseOdev);
  if Gl.prcom
  then begin
    write(Odev,#15 );
    writeln('this is true');
  end;
  alldone := false;
  prin_num := 1;
  clrscr;
  gotoxy(1,1);
  textcolor(Gl.SubMenuHeadingColor);
  writeln(' Output Menu');
  writeln;
  textcolor(Gl.SubMenuItemColor);

  writeln(' Decision matrix');
  writeln(' Selected alternatives');
  writeln(' Quit - return to main menu');
  repeat
    lookmenu(3,prin_num,0,2,'DSQ');
    window(1,10,80,25);
    gotoxy(1,1);
    clrscr;
    case prin_num of
      1 : begin
            Writeln(Odev,'The Decision Matrix');
            me2(decmat^,true);
          end;
      2 : begin
            Writeln(Odev,'The Non-Dominated Alternatives');
            me2(decmat^,false);
          end;
      3 : alldone:=true;
    end;
  window(1,1,80,25);
  until alldone;
  Flush(Odev);
  Screen_out(G_CloseOdev);
end;


File: getfname.inc
{
  Display messages to operator on last two lines of screen
  Functions properly only if screen is fullsized
```

```
}
procedure Msg2(S: AnyString);
begin
  Gotoxy(1,25);
  clreol;
  write(S);
end;

procedure Msg(S: AnyString);
begin
  Gotoxy(1,24);
  clreol;
  write(S);
end;

procedure WaitMsg(S: AnyString);
begin
  Msg(S);
  Msg2('Press any key to continue . . .');
  repeat until keypressed;
  Msg('');
  Msg2('');
end;

{*********************************************************************}
{*  File handeling subroutines                                       *}
{*********************************************************************}

function Exist(FileN: AnyString): boolean;
var F: file;
begin
   {$I-}
   assign(F,FileN);
   reset(F);
   close(F);
   {$I+}
   if IOResult<>0 then Exist:=false
   else Exist:=true;
end;

{
Reads a filename.  Doesnt allow numbers in filename.  Adds the default
extension if none is given.
}

procedure GetFileName(var Line: AnyString; FileType:String3);
var
  valid_name : boolean;
  j          : integer;
begin
  repeat
    readln(Line);
```

```
    valid_name := true;
    for j := 1 to length(Line) do begin
      if not (Upcase(Line[j]) in
['A'..'Z',':','.','0'..'9','$','@','#','_'])
      then valid_name := false;
      end;
    if not valid_name then Msg2('Invalid file name. ');
  until valid_name;
  if ( Line <> '' ) and ( pos('.',Line) = 0 )
  then Line:=Line+'.'+FileType;
end;


{$v-}
{ This program will search a directory for a matching file name and
  then print it. Other information is available in the record 'dta_def'.
  Original code:

          Jim Holtman
          35 Dogwood Trail
          Randolph, NJ 07869
          (201) 361-3395

  Modifications by: Dan
}

type
  regset = record
    ax,bx,cx,dx,bp,si,di,ds,es,flags : integer;
  end;
  fname = array[1..80] of char;
  str80 = string[80];
  dta_def = record
  filler : array[1..21] of byte;
    attribute : byte;
    file_time : integer;
    file_date : integer;
    file_size : array[1..2] of integer;
    file_name : fname;
  end;

const
  carry = 1;
  directory = $10;


procedure listfiles(pattern:str80);
var
  dta : dta_def;
  param : regset;
  s_string : string[70];
  r1,r2 : real;
  dta_save : array[1..2] of integer;
```

```
  ud_pos, lr_pos : integer;

function pack_name(var a1;
         size : integer) : str80;
var
  i : integer;
  b : str80;
  a : array[1..1000] of char absolute a1;
begin
  i := 1;
  b := '';
  while (a[i]<>chr(0)) and (i <= size) do begin
    b := b+a[i];
    i := i+1;
  end;
  pack_name := b;
end;

begin {listfiles}
  ud_pos := 15;
  lr_pos :=  1;
  pattern := pattern+chr(0);
  with param,dta do begin
    ax := $2F00;           {get DTA}
    msdos(param);
    dta_save[1] := es;
    dta_save[2] := bx;
    ax := $1A00;           {set DTA}
    ds := seg(dta);
    dx := ofs(dta);
    msdos(param);
    ds := seg(pattern[1]);
    dx := ofs(pattern[1]);
    ax := $4E00;           {find 1st}
    cx := $FF;
    msdos(param);
    while (flags and carry) = 0 do begin
      s_string := pack_name(file_name,sizeof(file_name));
      if ((attribute and directory) <> 0) and (s_string <> '.') and
(s_string <> '..')
      then begin end
      else begin
        gotoxy(lr_pos,ud_pos);
        write(s_string,'':(16-length(s_string)));
        ud_pos := ud_pos+1;
        if ud_pos = 22
        then begin
          ud_pos := 15;
          lr_pos := lr_pos + 16;
        end;
      end;
```

```
      ax := $4F00;        {get next}
      msdos(param);
    end;
    ax := $1A00;          {set DTA}
    ds := dta_save[1];
    dx := dta_save[2];
    msdos(param);
    end;
end;

File: getyes.inc
function Getyes: boolean;
var
  Ch : char;
begin
  repeat
    read(Kbd,Ch);
    Ch := upcase(Ch);
  until Ch in ['Y','N',^M];
  Getyes := (Ch = 'Y') or (Ch = ^M);
end;

File: inpstrng.inc
procedure Beep;
begin
  Write(^G);
end;

{$V-}

function ConstStr(C : Char; N : Integer) : Str80;
var
  S : string[80];
begin
  if N < 0 then
    N := 0;
  S[0] := Chr(N);
  FillChar(S[1],N,C);
  ConstStr := S;
end;

(*  UpcaseStr converts a string to upper case *)

function UpcaseStr(S : Str80) : Str80;
var
  P : Integer;
begin
  for P := 1 to Length(S) do
    S[P] := Upcase(S[P]);
  UpcaseStr := S;
end;
```

```
procedure InputStr(var S     : AnyString;
                       L,X,Y : Integer;
                       Term  : CharSet;
                   var TC    : Char;
                   var insertOn : boolean);
var dx, tl : integer;
procedure writeModes;
begin
  textcolor(Gl.EditNameColor);
  gotoXY(1,23);clreol;
  Writeln(' ^Y  - delete line    ¦ Pg Up - Screen Up   ¦ Home   - Go To
line');
  Writeln(' Ins - toggles insert ¦ Pg Dn - Screen Down ¦ Return - Quit
Edit');
  if insertOn then Write('Insert On ') else Write('Insert Off');
  textcolor(Gl.EditItemColor);
end;
const
  UnderScore  =  '_';
var
  P : Integer;
  Ch : Char;
begin
  WriteModes;
  GotoXY(X,Y); Write(S,ConstStr(UnderScore,L - Length(S)));
  P := 0;
  repeat
    GotoXY(X + P,Y); GetKey(Ch);
    case Ch of
      #32..#126 : if P < L
                  then begin
                    if insertOn
                    then begin
                      if Length(S) = L then Delete(S,L,1);
                      P := P + 1;
                      Insert(Ch,S,P);
                      Write(Copy(S,P,L));
                    end
                    else begin
                      if (Length(S) = P)
                      then begin
                        S := S + Ch;
                        P := P + 1;
                      end
                      else begin
                        P := P + 1;
                        S[P] := Ch;
                      end;
                      Write(Ch);
                    end
                  end
                  else Beep;
      ^S        : if P > 0 then
                    P := P - 1
```

```
                    else Beep;
        ^D          : if P < Length(S) then
                        P := P + 1
                      else Beep;
        ^A          : P := 0;
        ^F          : P := Length(S);
        ^G          : if P < Length(S) then
                      begin
                        Delete(S,P + 1,1);
                        Write(Copy(S,P + 1,L),UnderScore);
                      end;
        ^H,#127     : if P > 0 then
                      begin
                        Delete(S,P,1);
                        Write(^H,Copy(S,P,L),UnderScore);
                        P := P - 1;
                      end
                      else Beep;
{
        ^T          : if P < Length(S) then
                      begin
                        dx := 0;
                        tL := P-Length(S);
                        repeat dx := dx + 1 until (S[P+dx] =  ' ') or (dx = tL);
                        if dx <> tL
                        then repeat dx := dx + 1 until (S[P+dx] <> ' ') or (dx =
tL);
                        Delete(S,P+1,dx-1);
                        GotoXY(X,Y); Write(S,ConstStr(UnderScore,L -
Length(S)));
                      end
                      else Beep;
}
        ^Y          : begin
                        Write(ConstStr(UnderScore,Length(S) - P));
                        Delete(S,P + 1,L);
                      end;
        ^B          : begin
                        insertOn := not insertOn;
                        WriteModes;
                      end;
      else
        if not (Ch in Term) then Beep;
      end;  {of case}
    until Ch in Term;
    P := Length(S);
    GotoXY(X + P,Y);
    Write(' ':L - P);
    TC := Ch;
end;

File: scrnprtr.inc
procedure screen_out(var first:boolean);
```

```
begin
  if not first then Close(Odev) else first := true;
  G_owidth:=G_scwidth;
  assign(Odev,'con:');
  rewrite(Odev);
end;

procedure printer_out(var first: boolean);
begin
  if not first then Close(Odev) else first := true;
  G_owidth:=Gl.pwidth;
  assign(Odev,'lst:');
  rewrite(Odev);
end;

procedure Disk_out(var first:boolean);
var
  ok : boolean;
  Filename : AnyString;
  ch : char;
begin
  repeat
    Msg('Enter Output filename  ');
    GetFileName(Filename,'prn');
    ok := true;
    if exist(FileName)
    then begin
      Msg2('File already exists. Replace (Y/N) ');
      repeat read(Kbd,ch); ch := Upcase(ch); until ch in ['Y','N'];
      if ch ='Y' then ok := true else ok := false;
      end;
  until ok;
  if FileName<>''
  then begin
    if not first then Close(Odev) else first := true;
    G_owidth:=Gl.pwidth;
    assign(Odev,Filename);
    rewrite(Odev);
  end;
end;

Procedure OutputDevice(var first : boolean);
begin
  Msg('Output to (P)rinter, (S)creen, or (D)isk?');
  repeat read(Kbd,Ch); Ch := Upcase(Ch); until Ch in ['P','S','D'];
  case Ch of
    'P': printer_out(first);
    'S': screen_out(first);
    'D': Disk_out(first);
  end;
end;
```

```
File: namein.inc
procedure EditAttributeNames(var NameList: satbvect;
                             var numNames: integer;
                                 maxNames: integer;
                                 NameLength : integer);
const
  lines = 20;
var
  x,y : integer;
  g,h : integer;
  first,last: integer;
  oldFirst, oldLast : integer;
  ch,ch2 : char;
  curr : integer;


procedure displaylist(here, there : integer);
var g : integer;
begin
  clrscr;
  gotoXY(1,1);
  for g := here to there do
  begin
    writeln(NameList[g]);
  end;
end;

procedure boxrange(var top, bottom: integer);
begin
  if top  < 1 then top  := 1;
  bottom := top  + lines - 1;
  if bottom > numNames
  then begin
    bottom := numNames;
    top    := bottom - lines +1;
    end;
  if top  < 1 then top  := 1;
end;

procedure cursor_range;
begin
  if y <  1          then begin y := 1;     first := first - 1; end;
  if y >  lines      then begin y := lines; first := first + 1; end;
  if y >  numNames   then       y := numNames;
end;


procedure echo;
begin
  cursor_range;
  boxrange(first, last);
  if not (oldFirst = first)
```

```
    then begin
      displaylist(first,last);
      oldFirst := first;
    end;
    curr := y + first - 1;
  end;

  procedure goto_box_pos;
  var
    pos : integer;
  begin
    Msg2('Goto Alternative Number: ');
    limitin(pos,numNames,1);
    Msg2('');
    first := pos;
    echo;
    y := pos - first + 1;
  end;

  procedure deleteOption(d:integer; var num: integer);
  var h,j : integer;
  begin
    for h := d to num do NameList[h] := NameList[h+1];


  {write to delete out of decmat}

    for j := 1 to altnum do
      for h := d to num-1 do
        decmat^[j,h] := decmat^[j,h+1];

    num := num - 1;

    oldFIrst := 0;
    echo;

  end;

  begin
    oldFirst := 0;
    first := 1;
    y := 1;
    x := 1;
    echo;
    repeat

  InputStr(NameList[curr],NameLength,x,y,[^E,^X,^M,^Y,^R,^C,^A],ch,G_InsertMode
  On);
      case ch of
        ^E : y := y-1;
        ^X : y := y+1;
        ^M : y := y+1;
```

```
      ^Y : deleteOption(curr,numNames);
      ^A : goto_box_pos;
      ^R : first := first - 19;
      ^C : first := first + 19;
    end;
    echo;
  gotoXY(x,y);
  until (ch = ^M) or (numNames = 0);
  enteredAttributeNames := (numNames <> 0);
end;

procedure EnterAlternativeNames(var NameList: Saltvect;
                                var numNames: integer;
                                    maxNames: integer;
                                    nameLength : integer);
var
  x, y : integer;
  oldNumNames : integer;
procedure displayform;
begin
  clrscr;
  gotoxy(1,1);
  Writeln('Entering Alternative Names');
  Writeln;
  Writeln('Alternative # ');
  Writeln;
  Writeln('Name     :');
end;
procedure setform(num: integer);
begin
  gotoxy(17,3);write(num);
  gotoxy(x,y);
  clreol;
end;
begin
  oldNumNames := numNames;
  x := 12;
  y := 5;
  textcolor(Gl.EditHeadingColor);
  displayform;
  textcolor(Gl.EditItemColor);
  repeat
    numNames := numNames + 1;
    setform(numNames);
    InputStr(NameList[numNames],NameLength,x,y,[^M],ch,G_InsertModeOn);
  until (NameList[numNames] = '') or ((numNames) >= maxNames);
  if ((numNames) >= maxNames)
  then WaitMsg('Maximum number of names have been entered');
  if (NameList[numNames] = '') then numNames := numNames - 1;
  if numNames <> 0 then enteredAlternativeNames := true;
{ if DM retypes all prev data when entering new expert keep this line.
  if oldNumNames <> numNames then enteredAllocations := false;
```

```
}
end;

procedure EnterAttributeNames(var NameList: satbvect;
                              var numNames: integer;
                                  maxNames: integer;
                                  NameLength : integer);
var
  x, y : integer;
  oldNumNames : integer;
procedure displayform;
begin
  clrscr;
  gotoxy(1,1);
  Writeln('Entering Attribute Names');
  Writeln;
  Writeln('Attribute # ');
  Writeln;
  Writeln('Name     :');
end;
procedure setform(num: integer);
begin
  gotoxy(15,3);write(num);
  gotoxy(x,y);
  clreol;
end;
begin
  textcolor(G1.EditHeadingColor);
  displayform;
  textcolor(G1.EditItemColor);
  x := 12;
  y := 5;
  oldNumNames := numNames;
  repeat
    numNames := numNames + 1;
    setform(numNames);
    InputStr(NameList[numNames],NameLength,x,y,[^M],ch,G_InsertModeOn);
  until (NameList[numNames] = '') or ((numNames) >= maxNames);
  if ((numNames) >= maxNames)
  then WaitMsg('Maximum number of names have been entered');
  if (NameList[numNames] = '') then numNames := numNames - 1;
  if numNames <> 0 then enteredattributeNames := true;
{
  if oldNumNames <> numNames then enteredAllocations := false;
}
end;

procedure EditAlternativeNames(var NameList: saltvect;
                               var numNames: integer;
                                   maxNames: integer;
                                   NameLength : integer);
const
```

```
    lines = 20;
  var
    x,y : integer;
    g,h : integer;
    first,last: integer;
    oldFirst, oldLast : integer;
    ch,ch2 : char;
    curr : integer;


  procedure displaylist(here, there : integer);
  var g : integer;
  begin
    clrscr;
    gotoXY(1,1);
    for g := here to there do
    begin
      writeln(NameList[g]);
    end;
  end;

  procedure boxrange(var top, bottom: integer);
  begin
    if top  < 1 then top  := 1;
    bottom := top  + lines - 1;
    if bottom > numNames
    then begin
      bottom := numNames;
      top    := bottom - lines +1;
      end;
    if top  < 1 then top  := 1;
  end;

  procedure cursor_range;
  begin
    if y <  1         then begin y := 1;     first := first - 1; end;
    if y >  lines     then begin y := lines; first := first + 1; end;
    if y >  numNames  then       y := numNames;
  end;


  procedure echo;
  begin
    cursor_range;
    boxrange(first, last);
    if not (oldFirst = first)
    then begin
      displaylist(first,last);
      oldFirst := first;
    end;
    curr := y + first - 1;
  end;
```

```
procedure goto_box_pos;
var
  pos : integer;
begin
  Msg2('Goto Alternative Number: ');
  limitin(pos,numNames,1);
  Msg2('');
  first := pos;
  echo;
  y := pos - first + 1;
end;

procedure deleteOption(d:integer; var num: integer);
var h,j : integer;
begin
  for h := d to num do NameList[h] := NameList[h+1];


{write to delete out of decmat}

  for j := 1 to atbnum do
    for h := d to num-1 do
      decmat^[h,j] := decmat^[h+1,j];

  num := num - 1;

  oldFIrst := 0;
  echo;

end;

begin
  oldFirst := 0;
  first := 1;
  y := 1;
  x := 1;
  echo;
  repeat

InputStr(NameList[curr],NameLength,x,y,[^E,^X,^M,^Y,^R,^C,^A],ch,G_InsertMode
On);
    case ch of
      ^E : y := y-1;
      ^X : y := y+1;
      ^M : y := y+1;
      ^Y : deleteOption(curr,numNames);
      ^A : goto_box_pos;
      ^R : first := first - 19;
      ^C : first := first + 19;
    end;
    echo;
```

```
  gotoXY(x,y);
  until (ch = ^M) or (numNames = 0);
  enteredAlternativeNames := (numNames <> 0);
end;

File: numeric.inc
{
 The following procedures are used to verify that numbers are entered
 when requested.  The Turbo compiler dies when alphabetic characters are
 entered when numeric values are requested.  These procedures read in
 all input as character strings and convert to real or integer values.

del_leading_blanks(var s:anystring);
    deletes characters from the start of string until number is found
del_following_blanks(var s:anystring);
    deletes characters starting at the first delimiter (space or
    comma) after the number found by del_leading_blanks
read_int(var inval : integer; var good : boolean);
    reads a string, converts to integer, sets good flag.
read_real(var inval : real; var good : boolean);
    same as read_int except real
str_int(var temp_str: anystring; var inval : integer; var good : boolean);
    accepts as input string previously read, converts first number
    to integer, deletes number found from string, sets good flag.
str_real(var temp_str: anystring; var inval : real; var good : boolean);
    same as str_int except real
read_row_col_var(var int1, int2: integer; var real1: real; good: boolean);
    reads a string, returns two integers, and one real, and good flag
str_num(var istr     : anystring;
  This is the procedure that does all the work.  It is called by
  all the above procedures.
  istr     : The input string
  inta     : integer value returned
  reala    : real value returned
  int_type : reading a real or integer
  valid    : is there a valid number in the string
del_leading_blanks(var s:anystring);
    deletes characters from the start of string until number is found
del_following_blanks(var s:anystring);
    deletes characters starting at the first delimiter (space or
    comma) after the number found by del_leading_blanks
}

procedure str_num(var istr     : anystring;
                  var inta     : integer;
                  var reala    : real;
                      int_type : boolean;
                 var valid    : boolean);
var
  numbers  : set of char;
  delimiters : set of char;
  code     : integer;
```

```
    tempistr : anystring;

  procedure del_leading_blanks(var s:anystring);
  var
    ch      : char;
  begin
    repeat
      ch := s[1];
      if not (ch in numbers)
      then delete(s,1,1);
    until ( ch in numbers ) or ( length(s)=0 ) ;
  end;

  procedure del_following_blanks(var s:anystring);
  var
    ch : char;
    position : integer;
    num_ch_to_del : integer;
  begin
    s := s + ' ';                {add a blank to end of string in }
    position := 0;               {case it ends with a number      }
    repeat
      position := position + 1;
      ch := s[position];
    until ch in delimiters;
    num_ch_to_del := length(s)-position+1;
    delete(s,position,num_ch_to_del);
  end;

  begin
    numbers:=['0','1','2','3','4','5','6','7','8','9','.','+','-'];
    delimiters:=[' ',','];  { space and comma }
    inta:=0;
    reala:=0;
    valid := true;
    del_leading_blanks(istr);
    if length(istr) = 0
    then valid := false
    else begin
      tempistr := istr;
      del_following_blanks(tempistr);
      if int_type
      then val(tempistr,inta,code)
      else val(tempistr,reala,code);
      if code <> 0 then valid := false;
      delete(istr,1,length(tempistr));
    end;
  end;

  procedure read_int(var inval : integer; var good : boolean);
  var
    dummy    : real;
```

```
  temp_str : anystring;
begin
  read(temp_str);
  str_num(temp_str,inval,dummy,true, good);
end;

procedure read_real(var inval : real; var good : boolean);
var
  dummy : integer;
  temp_str : anystring;
begin
  read(temp_str);
  str_num(temp_str,dummy,inval,false,good);
end;

procedure str_int(var temp_str: anystring; var inval : integer; var good :
boolean);
var
  dummy : real;
begin
  str_num(temp_str,inval,dummy,true, good);
end;

procedure str_real(var temp_str: anystring; var inval : real; var good :
boolean);
var
  dummy : integer;
begin
  str_num(temp_str,dummy,inval,false,good);
end;

File config.inc
{$R-}
type
  ColorRange = 0..15;
  configtype = record
    MainMenuHeadingColor: ColorRange;
    MainMenuItemColor   : ColorRange;
    SubMenuHeadingColor : ColorRange;
    SubMenuItemColor    : ColorRange;
    EditHeadingColor    : ColorRange;
    EditItemColor       : ColorRange;
    EditNameColor       : ColorRange;
    EditRHSColor        : ColorRange;
    EditRelationColor   : ColorRange;
    EditCursorOffColor  : ColorRange;
    EditCursorOnColor   : ColorRange;
    pwidth              : integer;
    fwidth              : integer;
    decdis              : integer;
    decprt              : integer;
    prcom               : boolean;
```

```
  end;

var
  configfile  : file of configtype;
  Gl          : configtype;
  Ch          : char;

procedure Default;
begin
  with Gl do begin
    MainMenuHeadingColor:=Red;
    MainMenuItemColor   :=Yellow;
    SubMenuHeadingColor :=Yellow;
    SubMenuItemColor    :=Brown;
    EditHeadingColor    :=Red;
    EditItemColor       :=Yellow;
    EditNameColor       :=Red;
    EditRHSColor        :=Brown;
    EditRelationColor   :=Green;
    EditCursorOffColor  :=Green;
    EditCursorOnColor   :=Red;
    pwidth              :=80;
    fwidth              := 7;
    decdis              := 2;
    decprt              := 2;
    prcom               := false;
  end
end;

procedure LoadConfig;
begin
  if (not exist('config.cnf'))
  then Default
  else begin
    Assign(configfile,'config.cnf');
    Reset(configfile);
    Read(configfile,Gl);
    Close(configfile);
  end;
end;


Procedure Config;
var
  j : integer;
  C : array[1..11] of record
                        color : colorrange;
                        use   : string[30];
                      end;

Procedure Init;
begin
```

```
  C[1].color := G1.MainMenuHeadingColor;
  C[1].use   :='Main Menu Heading Color';
  C[2].color :=(G1.MainMenuItemColor);
  C[2].use   :=('Main Menu Item Color    ');
  C[3].color :=(G1.SubMenuHeadingColor);
  C[3].use   :=('Sub Menu Heading Color ');
  C[4].color :=(G1.SubMenuItemColor);
  C[4].use   :=('Sub Menu Item Color     ');
  C[5].color :=(G1.EditHeadingColor);
  C[5].use   :=('Edit Heading Color      ');
  C[6].color :=(G1.EditItemColor);
  C[6].use   :=('Edit Item Color         ');
  C[7].color :=(G1.EditNameColor);
  C[7].use   :=('Edit Name Color         ');
  C[8].color :=(G1.EditRHSColor);
  C[8].use   :=('Edit RHS Color          ');
  C[9].color :=(G1.EditRelationColor);
  C[9].use   :=('Edit Relation Color     ');
  C[10].color :=(G1.EditCursorOffColor);
  C[10].use   :=('Edit Cursor Color / Select');
  C[11].color :=(G1.EditCursorOnColor);
  C[11].use   :=('Edit Cursor Color / Change');
end;

procedure Init2;
begin
  G1.MainMenuHeadingColor :=C[1].color;
  G1.MainMenuItemColor    :=C[2].color;
  G1.SubMenuHeadingColor  :=C[3].color;
  G1.SubMenuItemColor     :=C[4].color;
  G1.EditHeadingColor     :=C[5].color;
  G1.EditItemColor        :=C[6].color;
  G1.EditNameColor        :=C[7].color;
  G1.EditRHSColor         :=C[8].color;
  G1.EditRelationColor    :=C[9].color;
  G1.EditCursorOffColor   :=C[10].color;
  G1.EditCursorOnColor    :=C[11].color;
end;

procedure printG;
begin
  with G1 do Writeln(
    MainMenuHeadingColor:3,
    MainMenuItemColor   :3,
    SubMenuHeadingColor :3,
    SubMenuItemColor    :3,
    EditHeadingColor    :3,
    EditItemColor       :3,
    EditNameColor       :3,
    EditRHSColor        :3,
    EditRelationColor   :3,
    EditCursorOffColor  :3,
```

```
    EditCursorOnColor   :3,
    pwidth              :3,
    fwidth              :3,
    decdis              :3,
    decprt              :3,
    prcom               :5);
end;

procedure SelectGColor;
var
  j : integer;
function done(var color:ColorRange):boolean;
var ch : char;
begin
    done := false;
    read(Kbd,ch);
    if keypressed then read(Kbd,ch);
    case upcase(ch) of
     'H':if j > 1 then begin j := j - 2; done := true; end;
     'K':color := color - 1;
     'M':color := color + 1;
     'P':done := true;
     ^M :begin j := 11; done := true; end;
    end; {case}
    if color =  0 then color := 15;
    if color > 15 then color :=  0;
end;

begin
  clrscr;
  gotoxy(1,1);
  textcolor(G1.SubMenuHeadingColor);
  Writeln('  Configure Menu');
  gotoxy(1,3);
  for j := 1 to 11 do begin
    textcolor(C[j].color);
    Writeln(' ',C[j].use);
  end;
  j := 0;
  repeat
    j := j + 1;
    repeat
      gotoxy(1,2+j);
      textcolor(C[j].color);
      Writeln(' ',C[j].use);
      gotoxy(1,2+j);
    until done(C[j].color);
  until j = 11;
end;

procedure SelectGPrinter;
var
```

```
  vin : integer;
  ch  : char;
  ok  : boolean;

procedure limitin(var out:integer; min, max: integer; var good : boolean);
var
  s : string[5];
  p : integer;
begin
  repeat
    out :=0;
    good := true;
    read(s);
    gotoxy(1,25);
    clreol;
    if length(s) <> 0
    then begin
      val(s,out,p);
      good := (p = 0);
      if not good or (good and ((out < min) or (out > max)))
      then begin
        good := false;
        gotoxy(1,25);
        clreol;
        Write('Expecting Numeric Value between ',min:4,' and ',max:4,' ');
      end; {not good or ( . . . )
    end; { if length <> 0 }
  until good;
  if length(s) = 0 then good := false;
end; {procedure limitin}

begin
    textcolor(yellow);

    gotoxy(1,16);
    Write('Printer width          [',Gl.pwidth:5,']');
    limitin(vin,30,256,ok);
    if ok then Gl.pwidth := vin;

    gotoxy(1,17);
    Write('Field width            [',Gl.fwidth:5,']');
    limitin(vin,5,15,ok);
    if ok then Gl.fwidth := vin;

    gotoxy(1,18);
    Write('Decimal places/display [',Gl.decdis:5,']');
    limitin(vin,0,8,ok);
    if ok then Gl.decdis := vin;

    gotoxy(1,19);
    Write('Decimal places/print   [',Gl.decprt:5,']');
    limitin(vin,0,8,ok);
```

```
    if ok then Gl.decprt := vin;

    gotoxy(1,20);
    Write('Print commpressed      [',Gl.prcom:5,']');
    repeat Read(Kbd,Ch) until Upcase(Ch) in ['Y','N',^M,'T','F'];
    if Upcase(Ch) in ['Y','T'] then Gl.prcom := true;
    if Upcase(Ch) in ['N','F'] then Gl.prcom := false;
    Writeln(Gl.prcom);

end;

procedure SaveConfig;
begin
  Assign(configfile,'config.cnf');
  Rewrite(configfile);
  write(configfile,Gl);
  Close(configfile);
end;

begin (* config *)
  LoadConfig;

  repeat
    Init;
    SelectGColor;
    Init2;
    SelectGPrinter;
    gotoxy(1,24);
    Write('Are these values correct? (Y/N)');
    repeat Read(Kbd,Ch) until Upcase(Ch) in ['Y','N',^M];
  until not (Upcase(Ch) ='N');

  SaveConfig;
end;
{$R+}

File: exit.inc
procedure exxit(var done: boolean);
var Ch : char;
begin
  Msg('Exit from system? (Y/N)');
  Msg2('(Has data been saved?)');
  repeat read(Kbd,Ch); Ch := Upcase(Ch) until Ch in ['Y','N'];
  done := Ch = 'Y';
end;

File: menu.inc
const
  escape = #27;
type
  Menustring = string[10];
```

```
procedure IBMCh(var Ch: Char);
begin
  case Ch of
    'H': Ch:=^E; { Up arrow    }
    'P': Ch:=^X; { Down arrow  }
    'M': Ch:=^D; { Right arrow }
    'K': Ch:=^S; { Left arrow  }
    'G': Ch:=^A; { Home Key    }
    'O': Ch:=^F; { End Key     }
    'S': Ch:=^G; { Del Key     }
    'R': Ch:=^B; { Ins key     }
    'I': Ch:=^R; { Pg Up       }
    'Q': Ch:=^C; { Pg Dn       }
    'ä': Ch:=^R; { ^Pg Up      }
    'V': Ch:=^C; { ^Pg Dn      }

  end; { of case }
end; { of procedure IBMChar }

procedure GetKey(var choice:char);
var
  EscChoice : char;
begin
  EscChoice := chr(0);
  repeat until Keypressed;
  read(Kbo,Choice);
  if keypressed
  then begin
    read(Kbd,EscChoice);
    IbmCh(EscChoice);
    Choice := EscChoice;
  end;
end;

{Controls menu selection.  Moves cursor position by cursor arrow input.

num -- number of menu items.
ix -- returns number of menu item selected.
yofset and xofset -- define the upper left corner of the menu.
}

procedure GetKeys(var choice, EscChoice:char);
begin
  EscChoice := chr(0);
  repeat until Keypressed;
  read(Kbd,Choice);
  if keypressed then read(Kbd,EscChoice);
end;

procedure lookmenu(    num : integer;
                   var ix  : integer;
                       yofset, xofset: integer;
```

```
                        FirstLetter : MenuString);
 var
   ch1: char;
   ch2: char;
   FirstPos : integer;
 begin
   repeat
     FirstPos := 0;
     gotoxy(yofset+1,xofset+ix);
     GetKeys(Ch1,Ch2);
     if Ch1 = escape
     then begin
       case upcase(Ch2) of
         'G':ix:=1;
         'H':ix:=ix-1;
         'P':ix:=ix+1;
         'O':ix:=num;
       end;
       if ix < 1   then ix:=num;
       if ix > num then ix:=1;
     end
     else begin
       ch1 := upcase(ch1);
       FirstPos := pos(ch1,FirstLetter);
       if FirstPos <> 0 then ix := FirstPos;
       gotoxy(yofset+1,xofset+ix);
     end;
   until (ch1 =^M) or (FirstPos <> 0) (or (F_KeyPos <> 0));
end;

File eigen.inc
type
  cray = array[1..9] of real;
const
  cr : cray =( 0.00, 0.00, 0.58, 0.90, 1.12, 1.24, 1.32, 1.41, 1.49);

{calculates the DM weights from the pairwise comparicon matrix}
procedure calceigen(     mtrx: bmat;
                     var weight: weightArray;
                         size: integer;
                     var consty: real);
{sums the columns of the eigenvector }
procedure mcsum(mat:bmat; var  cvec:weightArray; size:integer);
var g, h: integer;
  begin
    for g:= 1 to size do begin
      cvec[g]:=0;
      for h:= 1 to size do begin
        cvec[g]:=cvec[g]+mat[h,g];
      end;
    end;
  end; {proc mcsum}
```

```
{sums the rows of the eigenvector }
procedure mrsum(mat:bmat; var cvec:weightArray; size:integer);
var g, h: integer;
  begin
    for g:= 1 to size do begin
      cvec[g]:=0;
      for h:= 1 to size do begin
        cvec[g]:=cvec[g]+mat[g,h];
      end;
    end;
  end; {proc mcsum}

var
  csum,
  r1sum,
  r2sum,
  r3sum  : weightArray;
  sum,
  lambda : real;
  c      : bmat;
  g,h    : integer;
{calculates the eigenvector. simplified form.}
begin  {calceigen }
    mcsum(mtrx,csum,size);
    for g:=1 to size do for h:=1 to size do c[g,h]:=mtrx[g,h]/csum[h];
    mrsum(c,r1sum,size);
    for g:=1 to size do r1sum[g]:=r1sum[g]/size;
    for g:=1 to size do for h:=1 to size do c[g,h]:=mtrx[g,h]*r1sum[h];
    mrsum(c,r2sum,size);
    for g:=1 to size do r3sum[g]:=r2sum[g]/r1sum[g];
    sum:=0;
    for g:=1 to size do sum:=sum+r3sum[g];
    lambda:=sum/size;
    if (size > 1)
    then Consty := (lambda-size)/(size-1);
    for g := 1 to size do weight[g]:=r1sum[g];
end;  {calceigen }


procedure MakeEigenScreen(Name: nameType);
const
  offset = 20;
  start  =  6;
begin
  clrscr;
  Textcolor(G1.EditRelationColor);
  gotoXY(offset,1);
  write('With Respect To ');
  Textcolor(G1.EditRHSColor);
  writeln(Name);
```

```
   Textcolor(Gl.EditRelationColor);
   gotoXY(offset,start);
   writeln('1 . . Equal Importance');
   gotoXY(offset,start+1);
   writeln('2');
   gotoXY(offset,start+2);
   writeln('3 . . Weak Importance');
   gotoXY(offset,start+3);
   writeln('4');
   gotoXY(offset,start+4);
   writeln( 5 . . Essential Importance );
   gotoXY(offset,start+5);
   writeln('6');
   gotoXY(offset,start+6);
   writeln('7 . . Demonstrated Importance');
   gotoXY(offset,start+7);
   writeln('8');
   gotoXY(offset,start+8);
   writeln('9 . . Absolute Importance');
end;

procedure eigein(    wrtName : nameType;
                     nameIn : namelist;
                 var mtrx: bmat;
                     size: integer);

var
  g,h : integer;
  ch : char;
  temp : real;
  Ia, Ib : integer; { items compared }
{controls selection of which attribute is more important, and how much}
procedure query(NameA, NameB: nameType; var pc:real);
const
  cursoron  = '=>';
  cursoroff = '  ';
var
  ix   : integer;
  iy   : integer;
  templ : integer;
  offset : integer;
begin
  ix:=1;
  iy:=1;
  if pc <> 0
  then begin
    if pc < 1
    then begin
       pc := 1/pc;
       iy := 2;
       end;
    ix:= round(pc);
```

```
    end;
    Textbackground(black);
    Textcolor(Gl.EditRHSColor);
    gotoxy(5,3);
    templ := length(NameA);
    Write(NameA:maxNameLength+1);
    if Upcase(NameA[templ]) = 'S'
    then begin
      write(' are     ');
      offset := 11+maxNameLength;
    end
    else begin
      write(' is      ');
      offset := 10+maxNameLength;
    end;
    write(' important than ');
    templ := length(NameB);
    write(NameB:templ);clreol;
    repeat
      gotoxy(offset,3);
      if iy=1
      then begin Textcolor(Gl.EditCursorOffColor);write('more'); end
      else begin Textcolor(Gl.EditCursorOnColor);write('less'); end;
      Textcolor(Gl.EditRelationColor);
      gotoxy(18,5+ix);
      write(cursoron);
      GetKey(Ch);
      gotoxy(18,5+ix);
      write(cursoroff);
      case ch of
        ^E      : ix:=ix-1;
        ^X      : ix:=ix+1;
        ^S      : iy:=1;
        ^D      : iy:=2;
      end;
      if ix < 1 then ix:=9;
      if ix > 9 then ix:=1;
    until upcase(ch) = ^M;
    pc:=ix;
    if iy=2
    then pc:=1/pc;
  end; {query}


  begin  {the eigenvector menu }
    MakeEigenScreen(wrtName);

    for g:=1 to size do
    begin
      for h:= 1 to size do
        if g<h
        then begin
```

```
        temp := mtrx[g,h];
        query(NameIn[g], NameIn[h], temp);
        mtrx[g,h]:=temp;
        mtrx[h,g]:=1/temp;
      end
      else if h=g
          then mtrx[g,h]:=1;
  end;
end;
```

File: echo.inc

```
{matrix echo . eigenvector matrix -used to print the matrix on the
printer and in the display mode of TOPSIS.  This proc is not used in the
EDIT mode.  Two separate procedures are used because the Decision Matrix
and the eigenvector matrix are not shaped the same. }

procedure me1(a: pointbmat);
var
 g,h,i,j : integer;
 k,l     : integer;
 pfields : integer;
begin
  pfields:=(G_owidth div Gl.fwidth)-2;
  h:= atbnum div pfields;
  if (atbnum MOD pfields) <> 0 then h:=h+1;
  for g:=1 to h do begin
    k:= pfields*(g-1)+1;
    l:= pfields*g;
    if l > atbnum then l:=atbnum;
    write(Odev,'            ');
    for i:= k to l do write(Odev,Copy(Concat(attb[i],'
'),1,Gl.fwidth),' ');
    writeln(Odev);
    for i:= 1 to atbnum do begin
      write(Odev,Copy(Concat(attb[i],'        '),1,Gl.fwidth),' ');
      for j:= k to l do write(Odev,a^[i,j]:Gl.fwidth:Gl.decdis,' ');
      writeln(Odev);
    end;
    writeln(Odev);
  end;
end;
```

File: attdsply.inc

```
{matrix echo . eigenvector matrix -used to print the matrix on the
printer and in the display mode of TOPSIS.  This proc is not used in the
EDIT mode.  Two separate procedures are used because the Decision Matrix
and the eigenvector matrix are not shaped the same. }

procedure me1(name: nameList; a: bmat; weight: weightArray; numNames:
integer);
var
 g,h,i,j : integer;
```

```
  k,l     : integer;
  pfields : integer;
begin
  pfields:=(G_owidth div Gl.fwidth)-4;
  h:= numNames div pfields;
  if (numNames MOD pfields) <> 0 then h:=h+1;
  for g:=1 to h do begin
    k:= pfields*(g-1)+1;
    l:= pfields*g;
    if l > numNames then l:=numNames;
    write(Odev,'':maxNameLength);
    write(Odev,'':Gl.fwidth);
    write(Odev,'':Gl.fwidth);
    for i:= k to l do write(Odev,Copy(Concat(name[i],'
'),1,Gl.fwidth),' ');
    writeln(Odev);
    for i:= 1 to numNames do begin
      write(Odev,Copy(Concat(name[i],bName),1,maxNameLength),' ');
      write(Odev,'( ',weight[i]:Gl.fwidth:Gl.decdis,' ) ');
      for j:= k to l do write(Odev,a[i,j]:Gl.fwidth:Gl.decdis,' ');
      writeln(Odev);
    end;
    writeln(Odev);
  end;
end;

procedure DisplayConsistencies;
var
  g, h : integer;
begin
  with GlData do
  begin
    writeln(Odev, 'Analytical hierarchial weighting method');
    writeln(Odev, 'Problem Title: ', Problem:maxNameLength ,
                  '  consistency: ', Consty:Gl.fwidth:Gl.decdis);
    Writeln(Odev);
    Writeln(Odev, ' name  ':maxNameLength, '  ',
                  ' weight ':Gl.fwidth);
    mel(name, Eig, weight, numItem);
    for g := 1 to numItem do
    begin
      Writeln(Odev,'Attribute name: ',
                   name[g]:maxNameLength, '  ',
                   'consistency: ',
                   Leaf[g].L_Consty:Gl.fwidth:Gl.decdis);
      if (GlData.Leaf[g].L_numItem <> 0)
      then with GlData.Leaf[g] do
      begin
        Writeln(Odev,' name  ':maxNameLength, '  ',
                     ' weight ':Gl.fwidth);
        mel(L_name, L_Eig, L_weight, L_numItem);
      end; { with }
```

```
        writeln(Odev);
      end;
     writeln(Odev);
     writeln(Odev);
    end; { with GlData do }
end;

procedure DisplayWeights;
var
  g, h : integer;
begin
  with GlData do
  begin
    writeln(Odev, 'Analytical hierarchial weighting method');
    writeln(Odev, 'Problem Title: ', Problem:maxNameLength);
    writeln(Odev);

    Writeln(Odev,''':maxNameLength, '  ','      ',
                    'local':Gl.fwidth,
                    'global':Gl.fwidth);
    Writeln(Odev,'Attribute name':maxNameLength, '  ','      ',
                    'weight':Gl.fwidth,
                    'weight':Gl.fwidth);
    for g := 1 to numItem do
    begin
    writeln(Odev,Copy(Concat(name[g],bName),1,maxNameLength),' ',
                    weight[g]:Gl.fwidth:Gl.decdis);
      if (GlData.Leaf[g].L_numItem <> 0)
      then with GlData.Leaf[g] do
        for h := 1 to L_numItem do
          writeln(Odev,'
',Copy(Concat(L_name[h],bName),1,maxNameLength),
                      ' ', L_weight[h]:Gl.fwidth:Gl.decdis,
                        L_lweight[h]:Gl.fwidth:Gl.decdis);
        writeln(Odev);
    end;
   writeln(Odev);
   writeln(Odev);
  end; { with GlData do }
end;


procedure Display;
var
  g,h : integer;
  ix : integer;
  procdone : boolean;

begin
  OutputDevice(G_CloseOdev);
  ix:=1;
  procdone:=false;
  clrscr;
```

```
    gotoxy(1,1);
    textcolor(Gl.SubMenuHeadingColor);
    Writeln(' Print Results');
    Writeln;
    textcolor(Gl.SubMenuItemColor);
    writeln(' Attribute names with weights');
    writeln(' Consistencies and pairwise comparison matrices');
    writeln(' Quit - return to main menu');
    repeat
      lookmenu(3,ix,0,2,'ACQ');
      textcolor(Gl.EditItemColor);
      window(1,8,80,25);
      gotoxy(1,1);
      clrscr;
      case ix of
        1 : DisplayWeights;
        2 : DisplayConsistencies;
        3 : procdone:=true;
      end;{ case}
    window(1,1,80,25);
    until procdone;
    Flush(Ddev);
    Screen_out(G_CloseDdev);
end;

procedure EnterNames;
var
  g : integer;

begin
  with GlData do
  begin
    { enter problem name}
    clrscr;
    gotoxy(1,3);
    Writeln('Enter the problem name:');
    {$V-}
    InputStr(Problem, MaxNameLength,
25,3,[^M,^E,^X,^A],Ch,InsertModeOn,false);
    {$V+}
    {enter first level of heirarchy }
    EnterLevelNames(Problem, Name, NumItem, maxLevelOne, MaxNameLength);
    {enter second level of heirarchy }
    for g := 1 to numItem do
    begin
{
      Writeln('Do you have attributes below: ',GlData.Name[g],'?');
      if getyes then
}
      with GlData.Leaf[g] do
      begin
        EnterLevelNames(GlData.Name[g], l_Name, l_NumItem, maxLevelTwo,
MaxNameLength);
```

```
      end; { withLeaf do }
    end; { for g := 1 to numItem }
    EnteredAttributeNames := (numItem > 0);
  end; { with GlData do }
end;

procedure EnterEigenVectors;
var
  g : integer;
begin
  with GlData do
  begin
    {enter first level of heirarchy }
    Eigein(Problem, Name, Eig, NumItem);
    {enter second level of heirarchy }
    for g := 1 to numItem do
    begin
      if (GlData.Leaf[g].L_numItem <> 0)
      then begin
{
        Writeln('Do you want to weight the attributes below:
',GlData.Name[g],'?');
        if getyes then
}
        with GlData.Leaf[g] do
        begin
          Eigein(GlData.Name[g], L_name, L_Eig, L_numItem);
          L_entered := true;
        end; { withLeaf do }
      end; { do the weight }
    end; { for g := 1 to numItem }
    entered := true;
    EnteredEigenVectors := True;
  end; { with GlData do }
end;


procedure kbdin;
var
  ix   : integer;
  done : boolean;
  g, h : integer;


begin { kbdin }
  ix:=1;
  done:=false;
  repeat
    clrscr;
    gotoxy(1,1);
    textcolor(Gl.SubMenuHeadingColor);
    Writeln(' Edit');
```

```
    Writeln;
    textcolor(Gl.SubMenuItemColor);
    writeln(' Attribute names');
    writeln(' Eigenvector weights');
    writeln(' Quit - return to main menu');
    lookmenu(3,ix,0,2,'AEQ');
    case ix of
      1 : if EnteredAttributeNames
          then EnterNames
          else EnterNames;
      2 : if EnteredEigenVectors
          then begin
            MSG('Re-enter Pairwise Comparisons? (Y/N)');
            repeat read(Kbd,Ch); Ch := Upcase(Ch) until Ch in ['Y','N'];
            if Ch = 'Y' then EnterEigenVectors;
          end
          else EnterEigenVectors;
      3 :done := true;
    end;
    ix := ix + 1;
  until done;
end;

procedure Start;
begin
  init(GlData);
  Msg(' Keyboard or File input? (K/F)');
  repeat Read(Kbd,Ch) until Upcase(Ch) in ['K','F'];
  Write(Upcase(Ch));
  if UpCase(Ch)='F'
  then load
  else Kbdin;
end;

File: attfile.inc
{
Saves dataset out to file.  If file already exists, prompts with a
"Replace Yes/No" message.
}
procedure Save;
var
  g,h,i: integer;
  ItemLength : integer;
  ok  : boolean;
begin
  repeat
    Msg('Save: Enter filename  ');
    GetFileName(Filename,'att');
    ok := true;
    if exist(FileName)
    then begin
      Msg2('File already exists. Replace (Y/N) ');
```

```
      repeat read(Kbd,Ch); Ch := Upcase(Ch); until Ch in ['Y','N'];
      if Ch ='Y' then ok := true else ok := false;
      end;
  until ok;
if FileName<>'' then
  begin
    Assign(attfile,FileName);
    Rewrite(attfile);
    write(attfile,GlData  );
    Close(attfile);
  end;
end;
{
Loads dataset from a file.  Prompts for a filename.  File has to exists
to be read.
}
procedure Load;
var g,h,i : integer;
begin
  listfiles('*.att');
  Msg('Load: Enter filename  ');
  GetFileName(Filename,'att');
  if (Filename<>'') then if (not exist(FileName)) then
  repeat
    Msg('File not Found: Enter another filename  ');
    GetFileName(Filename,'att');
  until exist(FileName) or (FileName='');
  if FileName<>'' then
  begin
    ClrScr;
    Msg('Please Wait. Reading File...');
    Assign(attfile,FileName);
    Reset(attfile);
    read(attfile,GlData);
    Close(attfile);
    EnteredAttributeNames := true;
    EnteredEigenVectors := true;
  end;
end; {of Procedure Load}

File: attname.inc
procedure Beep;
begin
  Write(^G);
end;

{$V-}

function ConstStr(C : Char; N : Integer) : nameType;
var
  S : NameType;
begin
```

```
    if N < 0 then
      N := 0;
    S[0] := Chr(N);
    FillChar(S[1],N,C);
    ConstStr := S;
  end;

  (*  UpcaseStr converts a string to upper case *)

  function UpcaseStr(S : nameType) : nameType;
  var
    P : Integer;
  begin
    for P := 1 to Length(S) do
      S[P] := Upcase(S[P]);
    UpcaseStr := S;
  end;

  procedure InputStr(var S     : NameType;
                         L,X,Y : Integer;
                         Term  : CharSet;
                     var TC    : Char;
                     var insertOn : boolean;
                         maxNumIn : boolean);

  var dx, t1 : integer;
  procedure writeModes;
  begin
    textcolor(G1.EditNameColor);
    gotoXY(1,23);clreol;
    Writeln(' ^Y  - delete line    |  Home   - Quit Edit');
    Writeln(' Ins - toggles insert |  Return - ');
    if insertOn then Write('Insert On ') else Write('Insert Off');
    if maxNumIn then Write('Maximum number of names have been entered')
    else clreol;
    textcolor(G1.EditItemColor);
  end;
  const
    UnderScore  =  '_';
  var
    P : Integer;
    Ch : Char;
  begin
    WriteModes;
    GotoXY(X,Y); Write(S,ConstStr(UnderScore,L - Length(S)));
    P := 0;
    repeat
      GotoXY(X + P,Y); GetKey(Ch);
      case Ch of
        #32..#126 : if P < L
                    then begin
                      if insertOn
```

```
                    then begin
                      if Length(S) = L then Delete(S,L,1);
                      P := P + 1;
                      Insert(Ch,S,P);
                      Write(Copy(S,P,L));
                    end
                    else begin
                      if (Length(S) = P)
                      then begin
                        S := S + Ch;
                        P := P + 1;
                      end
                      else begin
                        P := P + 1;
                        S[P] := Ch;
                      end;
                      Write(Ch);
                    end
                  end
                  else Beep;
    ^S          : if P > 0 then
                    P := P - 1
                  else Beep;
    ^D          : if P < Length(S) then
                    P := P + 1
                  else Beep;
    ^A          : P := 0;
    ^F          : P := Length(S);
    ^G          : if P < Length(S) then
                  begin
                    Delete(S,P + 1,1);
                    Write(Copy(S,P + 1,L),UnderScore);
                  end;
    ^H,#127     : if P > 0 then
                  begin
                    Delete(S,P,1);
                    Write(^H,Copy(S,P,L),UnderScore);
                    P := P - 1;
                  end
                  else Beep;
    ^Y          : begin
                    Write(ConstStr(UnderScore,Length(S) - P));
                    Delete(S,P + 1,L);
                  end;
    ^B          : begin
                    InsertOn := not InsertOn;
                    WriteModes;
                  end;
  else
    if not (Ch in Term) then Beep;
  end;  {of case}
until Ch in Term;
```

```
    P := Length(S);
    GotoXY(X + P,Y);
    Write('' :L - P);
    TC := Ch;
  end;


  procedure EnterLevelNames(    wrtName : nameType;
                            var namesIn : nameList;
                            var numNames: integer;
                                maxNames: integer;
                                nameLength : integer);
  var
    edNum : integer;
    tempName : nameType;
    Column, Line : integer;
    Message : boolean;

  procedure displayform(CharPos, LinePos: integer);
  var g : integer;
  begin
    clrscr;
    gotoxy(1,1);
    textcolor(Gl.EditHeadingColor);
    Writeln;
    Writeln;
    Write('Entering attributes below ');
    textcolor(Gl.EditRHSColor);
    Writeln(wrtName);
    Writeln;
    textcolor(Gl.EditItemColor);
    for g := 1 to numNames do
    begin
      gotoXY(CharPos, LinePos + g);
      clreol;
      write(namesIn[g]);
    end;
  end;

  begin
    Column := 12;
    Line   := 5;
    edNum  := 1;
    Message := false;
    displayform(Column, Line);
    textcolor(Gl.EditItemColor);
    repeat
      {$V-}
      InputStr(namesIn[edNum],NameLength,Column, Line+edNum,
  [^E,^X,^M,^A],ch,InsertModeOn,message);
      {$V+}
      if edNum > numNames then numNames := edNum;
```

```
    case ch of
      ^E : edNum := edNum-1;
      ^X : edNum := edNum+1;
      ^M : edNum := edNum+1;
    end;
    if edNum >  maxNames then edNum :=  1;
    if edNum <  1 then edNum :=  numNames;
    if ((numNames) >= maxNames) and not message
    then message := true;
  gotoXY(Line+edNum,Column);
  until ch = ^A;
  while (numNames > 0) and (namesIn[numNames] = '') do
    numNames := numNames - 1;
end;

File elect1.inc
{$R+,C-,V-}
{
  This program executes the ELECTRE algorithm.  This program is menu driven.
  The arrow keys on the numeric keypad will move the blinking cursor to
  the option you desire.  The return key, <ret>, then selects that option.
  Department of Industrial Engineering Kansas State University.
  The program was written by Dan McAnarney.
}
{ Copyright 1985, 1986 Version 2.04   }

program electre(input,output);
type
  cray = array[1..9] of real;
const
  altmax    =  101;         {Maximum number of alternatives }
  atbmax    =   25;         {Maximum number of attributes   }
  G_scwidth =   78;         {Screen width (or printer width }
  naltlen   =   30;         {fieldwidth for strings }
  natblen   =    9;         {fieldwidth for strings }
  lines     =   15;
  blank15   =   '';
  cr : cray =( 0.00, 0.00, 0.58, 0.90, 1.12, 1.24, 1.32, 1.41, 1.49);
type
  charset   = set of char;
  pointamat = ^amat;
  amat      = array [1..altmax,1..atbmax]of real;
  point8mat = ^amat;
  bmat      = array [1..atbmax,1..atbmax]of real;
  pointcdmat= ^cdmat;
  cdmat     = array [1..altmax,1..altmax]of real;
  pointAlt  = ^altvect;
  altvect   = array [1..altmax] of real;
  atbvect   = array [1..atbmax] of real;
  ivect     = array [1..atbmax] of integer;

  namealt   = string[naltlen];
```

```
    nameatb    = string[natblen];

    AnyString = string[70];
    String3   = string[3];
    saltvect  = array [1..altmax] of namealt;
    satbvect  = array [1..atbmax] of nameatb;

  var
    decmat     : pointamat;            {decision matrix}
    emat       : pointBmat;       {eigenvector matrix}
    altnum     : integer;         {number of alternatives}
    atbnum     : integer;         {number of attributes}
    dmwght     : atbvect;         {Decision Makers Weights}
    wnormat : pointamat;          {weighted normalized decision matrix}
    aggregate,
    concord,
    discord : pointcdmat;
    ConThreshold,
    DisThreshold : real;
    altn       : saltvect;        {labels for alternative names}
    attb       : satbvect;        {labels for attribute names }
    cost_benefit : ivect;      { 1-benefit, -1-cost}
    FileName   : string[70];
    Line       : string[100];
    mdmfile    : text;            {output disk file. save data }
    Odev       : text;            {Output device, 'lst:','scn:'}
    consty     : real;            {consistency of the eigenvector}
    G_owidth   : integer;         { output device width}
    G_CloseOdev     : boolean;
    entered_DM      : boolean;
    G_InsertModeOn : boolean;
    enteredAlternativeNames : boolean;
    enteredAttributeNames : boolean;


{$I common\numeric.inc}
{$I common\getfname.inc}
{$I common\menu.inc}
{$I common\config.inc}
{$I common\inpstrng.inc}
{$I common\scrnprtr.inc}
{$I mdmfile.inc}
{$I mdmechos.inc}


{when integer input variables are are known to be between two
predetermined values, this procedure is called.  For example, the number
of alternatives must be between 2 and altmax (currently 200), so limitin
is called with altnum as the variable returned, 200 and 2 as the limits}

procedure limitin(var inval:integer;max,min:integer);
var
```

```
  ok : boolean;
begin
  repeat
    read_int(inval,ok);
    if not ok or (inval>max) or (inval<min)
    then begin
      Msg2('Maximum and Minimum values are: ');
      write(max:5,min:5,'  ');
    end;
  until (inval>=min) and (inval<=max);
end;

{$I namein.inc}

{$I eleprint.inc }

{$I mdmedit.inc}

{$I mdmwght.inc}

{$I mdminput.inc}

{$I elealg.inc}

{$I common\exit.inc}

{initialize variables for startup of new problem}
procedure init;
var
  g,h : integer;
begin
  for g:= 1 to altmax do for h:= 1 to atbmax do decmat^[g,h]:=0;
  for g:= 1 to atbmax do for h:= 1 to atbmax do emat^[g,h]:=0;
  for g:= 1 to altmax do altn[g]:=blank15;
  for g:= 1 to atbmax do attb[g]:=blank15;
  for g:= 1 to atbmax do dmwght[g]:=0;
  for g:= 1 to atbmax do cost_benefit[g]:=1;
  altnum := 0;
  atbnum := 0;
  entered_DM     := false;
  enteredAlternativeNames := false;
  enteredAttributeNames := false;
end;

procedure start;
begin
  init;
  Msg(' Keyboard or File input? (K/F)');
  repeat Read(Kbd,Ch) until Upcase(Ch) in ['K','F'];
  Write(Upcase(Ch));
  if UpCase(Ch)='F'
  then load
```

```
  else kbdin;
end;

procedure main;
var
  menu_item : integer;
  all_done  : boolean;
  good_eqns : boolean;

procedure nomemory;
begin
  Msg('Not enough memory available to run program.');
  Halt;
end;

begin (* main *)
  {$I-}
  new(decmat);
  if IOResult<>0 then nomemory;
  new(emat);
  if IOResult<>0 then nomemory;
  new(wnormat);
  if IOResult<>0 then nomemory;
  new(concord);
  if IOResult<>0 then nomemory;
  new(discord);
  if IOResult<>0 then nomemory;
  new(aggregate);
  if IOResult<>0 then nomemory;
  {$I+}
  LoadConfig;
  G_InsertModeOn := true;
  G_CloseOdev    := true;
  all_done       := false;
  menu_item      := 1;
  init;
  screen_out(G_CloseOdev);
  repeat
    clrscr;
    textcolor(G1.MainMenuHeadingColor);
    gotoxy(1,1);
    Writeln('   Main Menu');
    Writeln;
    textcolor(G1.MainMenuItemColor);
    writeln(' Load new dataset');
    writeln(' Run ELECTRE');
    writeln(' Display results');
    writeln(' Edit input data');
    writeln(' Save current dataset');
    writeln(' Configure system');
    writeln(' eXit program');
    lookmenu(7,menu_item,0,2,'LRDESCX');
```

```
    case menu_item of
      1 :Start;
      2 :electre;
      3 :Display;
      4 :Kbdin;
      5 :save;
      6 :Config;
      7 :exxit(all_done);
    end;
  until all_done;
end;

begin
  main;
end.

File: eleprint.inc
{  matrix echo -used to print the matrix on the printer and in the
display mode of ELECTRE. This proc is not used in the EDIT mode.  }

procedure me3(var a: cdmat);
var
 g,h,i,j : integer;
 k,l     : integer;
 pfields : integer;
begin
  pfields:=((G_owidth-naltlen) div G1.fwidth)-1;
  if pfields <= 0 then pfields := 1;
  h:= altnum div pfields;
  if (altnum MOD pfields) <> 0 then h:=h+1;
  for g:=1 to h do begin
    k:= pfields*(g-1)+1;
    l:= pfields*g;
    if l > altnum then l:=altnum;
    write(Odev,'':naltlen,'   ');
    for i:= k to l do write(Odev,Copy(Concat(altn[i],'
'),1,G1.fwidth),' ');
    writeln(Odev);
    for i:= 1 to altnum do begin
      write(Odev,altn[i]:naltlen);
      write(Odev,'  ');
      for j:= k to l do write(Odev,a[i,j]:G1.fwidth:G1.decdis,' ');
      writeln(Odev);
    end;
    writeln(Odev);
  end;
end;
procedure Display;
var
  g,h : integer;
  n  : pointamat;           {normalized decision matrix}
  alldone   : boolean;
```

```
  prin_num  : integer;


begin
  OutputDevice(G_CloseOdev);
  alldone := false;
  prin_num := 1;
  clrscr;
  gotoxy(1,1);
  textcolor(Gl.SubMenuHeadingColor);
  writeln(' Output Menu');
  writeln;
  textcolor(Gl.SubMenuItemColor);

  writeln(' decision Matrix');
  writeln(' Pairwise comparison matrix');
  writeln(' dm Weights');
  writeln(' Thresholds for concordance/discordance');
  writeln(' Concordance dominance matrix');
  writeln(' Discordance dominance matrix');
  writeln(' Aggregate dominance matrix');
  writeln(' Quit - return to main menu');
  repeat
    lookmenu(8,prin_num,0,2,'MPWTCDAQ');
    window(1,11,80,25);
    gotoxy(1,1);
    clrscr;
    case prin_num of
      1 : begin
            Writeln(Odev,'The Decision Matrix');
            me2(decmat^);
          end;
      2 : begin
            Writeln(Odev,'The Eigenvector Pairwise comparisons');
            me1(emat);
          end;
      3 : begin
            writeln(Odev,'The DM subjective weights');
            ve2(dmwght);
          end;
      4 : begin
            write(Odev,'The concordance threshold: ');
            writeln(Odev,ConThreshold:Gl.fwidth:Gl.decdis);
            write(Odev,'The discordance threshold: ');
            writeln(Odev,DisThreshold:Gl.fwidth:Gl.decdis);
            writeln(Odev);
          end;
      5 : begin
            writeln(Odev,'The concordance matrix');
            me3(concord^);
          end;
      6 : begin
```

```
            writeln(Odev,'The discordance matrix');
            me3(discord^);
          end;
      7 : begin
            writeln(Odev,'The aggregate dominance matrix');
            me3(aggregate^);
          end;
      8 : alldone:=true;
    end;
  window(1,1,80,25);
  until alldone;
  Flush(Odev);
  Screen_out(G_CloseOdev);
end;

File: logoele.inc
Procedure logoele;
begin
   clrscr;
   Gotoxy(31, 1); Write('The ELECTRE Method');
   Gotoxy(39, 3); Write('for');
   Gotoxy(23, 5); Write('Multiple Attribute Decision Making');
   Gotoxy(34, 7); Write('programmed by:');
   Gotoxy(31, 9); Write('Daniel K. McAnarney');
   Gotoxy(21,11); Write('submitted in partial fulfillment of the');
   Gotoxy(27,13); Write('requirements of the degree');
   Gotoxy(32,15); Write('MASTER OF SCIENCE');
   Gotoxy(22,17); Write('Department of Industrial Engineering');
   Gotoxy(29,19); Write('Kansan State University');
   Gotoxy(32,21); Write('Manhattan, Kansas');
   Gotoxy(38,23); Write('1986');
   Gotoxy(20,25); Write('Copyright *c* 1985, 1986 by Dan McAnarney');
   delay(2000);
end;

File: hawmii.pas
{$V+}
{$R+}
const
  maxLevelOne   =    8;
  maxLevelTwo   =    8;
  maxAnyLevel   =    8;
  MaxNameLength =   15;
  testing       = TRUE;
  G_scwidth     =   79;
  bName         = '               ';

type
  String3   = string[3];
  CharSet   = set of char;
  AnyString = string[70];
  weightArray = array[1..maxAnyLevel] of real;
```

```
  nameType  = string[MaxNameLength];

  nameList  = array [1..maxAnyLevel] of nameType;
  bmat      = array[1..maxAnyLevel,1..maxAnyLevel] of real;
  LevelOneItemType =
  record
    Problem : nameType;
    entered : boolean;
    numItem : integer;
    Name    : nameList;
    Weight  : weightArray;
    Consty  : real;
    Eig     : bmat;
    Leaf        : array[1..maxLevelOne] of
                  record
                    l_Entered : boolean;
                    l_Numltem : integer;
                    l_Name    : nameList;
                    l_Weight  : weightArray;
                    l_Lweight : weightArray;
                    l_Consty  : real;
                    l_Eig     : bmat;
                  end;
  end;

var
  ix        : integer;        {menu item selector}
  FileName  : string[70];
  Line      : string[100];
  attfile   : file of LevelOneItemType;    {output disk file. save data }
  GlData    : LevelOneItemType;
  Odev          : text;           {Output device, 'lst:','scn:'}
  InsertModeOn : boolean;
  G_CloseOdev  : boolean;         {Does Odev need to be closed}
  G_owidth     : integer;         { output device width}
  EnteredAttributeNames : boolean;
  EnteredEigenVectors   : boolean;
  allDone               : boolean;

procedure init(var Item: LevelOneItemType);
var
  g, h, i : integer;
begin
  with Item do
  begin
    for i := 1 to MaxNameLength do Problem[i] := ' ';
    Problem := '';
    entered := false;
    numItem :=     0;
    Consty  :=     0;
    for g := 1 to maxLevelOne do
    begin
```

```
      for i := 1 to MaxNameLength do Name[g][i] := ' ';
      Name[g]    := '';
      Weight[g]  := 0;
    end;
    for g := 1 to maxLevelOne do
      for h := 1 to maxLevelOne do
        Eig[g,h]:= 0;
    for g := 1 to maxLevelOne do with Leaf[g] do
    begin
      l_entered := false;
      l_numItem := 0;
      l_Consty  := 0;
      for h := 1 to maxLevelOne do
      begin
        for i := 1 to MaxNameLength do l_Name[h][i] := ' ';
        l_Name[h] := '';
        l_Weight[h]  := 0;
        l_Lweight[h] := 0;
        for i := 1 to maxAnyLevel do l_Eig[h,i]:= 0;
      end; { for  h }
    end; { for g }
  end; { with }
end; {init}


{$I common\menu.inc}
{$I getyes.inc}
{$I common\getfname.inc}
{$I common\config.inc}
{$I attfile.inc}
{$I eigen.inc}
{$I namein.inc}
{$I common\scrnprtr.inc}
{$I attdsply.inc}
{$I hawmii.inc}

var
  g, h : integer;

begin (* main *)
  LoadConfig;
  G_CloseOdev  := true;
  InsertModeOn := true;
  screen_out(G_CloseOdev);
  clrscr;
  alldone:=false;
  ix:=1;
  repeat
    clrscr;
    gotoxy(1,1);
    textcolor(G1.MainMenuHeadingColor);
    Writeln(' MAIN MENU');
```

```
    Writeln;
    textcolor(Gl.MainMenuItemColor);
    writeln(' Load new dataset');
    writeln(' Run Heirarchial Additive Weighting Method');
    writeln(' Display Results');
    writeln(' Edit input data');
    writeln(' Save current dataset');
    writeln(' Configure system');
    writeln(' Quit program');
    lookmenu(7,ix,0,2,'LRDESCQ');
    case ix of
      1 : Start;
      2 : hawmll;
      3 : Display;
      4 : kbdin;
      5 : save;
      6 : Config;
      7 : alldone := true;
    end;
  until alldone;
end.

File: slecteig.pas
{$R+,C-,V-}
{
This program executes the Eigenvector selection algorithm.  This program
is menu drive The arrow keys on the numeric keypad will move the
blinking cursor to the option you desire.  The return key, <ret>, then
selects that option.  Department of Industrial Engineering Kansas State
University.  The program was written by Dan McAnarney.
}
{ Copyright 1986  Version 2.04   }

program selectattributes(input,output);
type
  cray = array[1..9] of real;
const
  atbmax    =   25;         {Maximum number of attributes   }
  G_scwidth =   80;         {Screen width (or printer width }
  natblen   =    9;         {fieldwidth for strings }
  lines     =   15;
  blank15   =   '';
  keepFactorA= 1.5;
  keepFactorB= 2.0;
  keepMinAtb =   4;
  keepMaxAtb =   6;
  cr : cray =( 0.00, 0.00, 0.58, 0.90, 1.12, 1.24, 1.32, 1.41, 1.49);
type
  saveType  = (all,some);
  charset   = set of char;
  pointBmat = ^bmat;
  bmat      = array [1..atbmax,1..atbmax]of real;
```

```
    atbvect    = array [1..atbmax] of real;
    atbvectboolean    = array [1..atbmax] of boolean;
    ivect      = array [1..atbmax] of integer;

    nameatb    = string[natblen];

    AnyString = string[70];
    String3   = string[3];
    satbvect  = array [1..atbmax] of nameatb;

var
    emat       : pointBmat;       {eigenvector matrix}
    nonDomAltnum : integer;       {number of non-dominated alternatives}
    atbnum     : integer;         {number of attributes}
    atbUsed    : atbvectBoolean;
    dmwght     : atbvect;         {Decision Makers Weights}
    attb       : satbvect;        {labels for attribute names }
    cost_benefit : ivect;      { 1-benefit, -1-cost}
    FileName   : string[70];
    Line       : string[100];
    domfile    : text;            {output disk file. save data }
    Odev       : text;            {Output device, 'lst:','scn:'}
    consty     : real;            {consistency of the eigenvector}
    G_owidth   : integer;         { output device width}
    G_CloseOdev        : boolean;
    G_InsertModeOn     : boolean;
    enteredAttributes : boolean;
    enteredWeights     : boolean;

{$I \common\numeric.inc}
{$I \common\getfname.inc}
{$I \common\menu.inc}
{$I \common\config.inc}
{$I \common\inpstrng.inc}
{$I \common\scrnprtr.inc}

{
Saves dataset out to file.  If file already exists, prompts with a
"Replace Yes/No" message.
}

procedure Save;
var
  g,h,i: integer;
  ItemLength : integer;
  ok  : boolean;
  SavedAlts : integer;
begin
  repeat
    Msg('Save: Enter filename  ');
    GetFileName(Filename,'eig');
    ok := true;
```

```
    if exist(FileName)
    then begin
      Msg2('File already exists. Replace (Y/N) ');
      repeat read(Kbd,ch); ch := Upcase(ch); until ch in ['Y','N'];
      if ch ='Y' then ok := true else ok := false;
      end;
  until ok;
if FileName<>'' then
  begin
    Assign(domfile,FileName);
    Rewrite(domfile);
    writeln(domfile,atbnum:15);
    for g:=1 to atbnum do for h:=1 to atbnum do writeln(domfile,emat^[g,h]);
    for h:=1 to atbnum do
    begin
      ItemLength := Length(attb[h]);
      writeln(domfile,attb[h]:ItemLength);
    end;
    for h:=1 to atbnum do
    writeln(domfile,dmwght[h]:15:10);
    for h:=1 to atbnum do
    writeln(domfile,cost_benefit[h]:5);

    h := 0; i := 0;
    if not enteredAttributes then h := -1;
    if not enteredWeights    then i := -1;
    writeln(domfile,h:5,i:5);
    Close(domfile);
  end;
end;


{
Loads dataset from a file.  Prompts for a filename.  File has to exists
to be read.
}

procedure Load;
var g,h : integer;
begin
  listfiles('*.eig');
  Msg('Load: Enter filename  ');
  GetFileName(Filename,'eig');
  if (Filename<>'') then if (not exist(FileName)) then
  repeat
    Msg('File not Found: Enter another filename  ');
    GetFileName(Filename,'eig');
  until exist(FileName) or (FileName='');
  if FileName<>'' then
  begin
    ClrScr;
    Msg('Please Wait. Reading File...');
```

```
      Assign(domfile,FileName);
      Reset(domfile);
      read(domfile,atbnum);
      for g:=1 to atbnum do for h:=1 to atbnum do read(domfile,emat^[g,h]);
      readln(domfile);
      for g:=1 to atbnum do
        readln(domfile,attb[g]);
      for g:=1 to atbnum do
        readln(domfile,dmwght[g]);
      for g:=1 to atbnum do
        readln(domfile,cost_benefit[g]);
      readln(domfile,h,g);
      enteredWeights := (g = 0);
      enteredAttributes := (h = 0);
      Close(domfile);
    end;
end; {of Procedure Load}


{matrix echo . eigenvector matrix -used to print the matrix on the
printer and in the display mode of TOPSIS.  This proc is not used in the
EDIT mode.  Two separate procedures are used because the Decision Matrix
and the eigenvector matrix are not shaped the same. }

procedure me1(a: pointbmat);
var
 g,h,i,j : integer;
 k,l     : integer;
 pfields : integer;
begin
  pfields:=(G_owidth div Gl.fwidth)-2;
  h:= atbnum div pfields;
  if (atbnum MOD pfields) <> 0 then h:=h+1;
  for g:=1 to h do begin
    k:= pfields*(g-1)+1;
    l:= pfields*g;
    if l > atbnum then l:=atbnum;
    write(Odev,'           ');
    for i:= k to l do write(Odev,Copy(Concat(attb[i],'
'),1,Gl.fwidth),' ');
    writeln(Odev);
    for i:= 1 to atbnum do begin
      write(Odev,Copy(Concat(attb[i],'       '),1,Gl.fwidth),' ');
      for j:= k to l do write(Odev,a^[i,j]:Gl.fwidth:Gl.decdis,' ');
      writeln(Odev);
    end;
    writeln(Odev);
  end;
end;


{vector echo . Echos vectors to printer or screen. }
```

```
procedure ve2(a: atbvect);
var
 g,h,i,j : integer;
 k,l     : integer;
 pfields : integer;
begin
  pfields:=(G_owidth div Gl.fwidth)-2;
  h:= atbnum div pfields;
  if (atbnum MOD pfields) <> 0 then h:=h+1;
  for g:=1 to h do begin
    k:= pfields*(g-1)+1;
    l:= pfields*g;
    if l > atbnum then l:=atbnum;
    for i:= k to l do write(Odev,Copy(Concat(attb[i],'
'),1,Gl.fwidth),' ');
    writeln(Odev);
    for i:= k to l do write(Odev,a[i]:Gl.fwidth:Gl.decdis,' ');
    writeln(Odev);
    writeln(Odev);
  end;
end;

procedure ve3(a: atbvectBoolean);
var
  g : integer;
begin
  for g:=1 to atbnum do
    if a[g] then write(Odev,Copy(Concat(attb[g],'        '),1,Gl.fwidth),'
');
  writeln(Odev);
end;

{This sorts the alternatives in order by closeness to the ideal
solution.  Alternative names are sorted with the closeness measures as a
matched pair.  The algorithm used is the shell-sort.  Not as fast as a
quicksort, but much less code.  Much faster than a bubble sort, with
about the same amount of code. }


procedure orderattributes(vec: atbvect; var list: ivect; length:integer);
var
  jump, m, n : integer;
  temp       : real;
  tempint    : integer;
  alldone    : boolean;
begin
  jump:=length;
  while jump > 1 do begin
    jump :=jump DIV 2;
    repeat
      alldone:=true;
      for m:=1 to length-jump do begin
```

```
        n:= m+jump;
        if vec[m] < vec[n]
        then begin
          temp:=vec[m];
          vec[m]:=vec[n];
          vec[n]:=temp;
          tempint:=list[m];
          list[m]:=list[n];
          list[n]:=tempint;
          alldone:=false;
        end;
      end; { for }
    until alldone;
  end; { while }
end; { sort }

procedure selectWeights(var using:atbvectBoolean);
var
  order : ivect;
  j : integer;
  keptPercent : real;

begin
  for j := 1 to atbmax do begin
    order[j] := j;
    using[j] := false;
  end;
  orderattributes(dmwght,order,atbnum);
  for j := 1 to keepMinAtb do
  begin
    using[order[j]]:= true;
    keptPercent := keptPercent + dmwght[order[j]];
  end;
  for j := keepMinAtb+1 to keepMaxAtb do
    if  (dmwght[order[j-1]]/dmwght[order[j]] < keepFactorA)
    and (keptPercent/(dmwght[order[j]]*(j-1)) < keepFactorB)
    then using[order[j]]:= true;
{
  for j := 1 to atbnum do write(order[j]:3,using[j]:6);
  writeln;
  writeln('waiting. . .');
  repeat until keypressed;
}
end;

procedure select;
begin
  if not enteredWeights
  then WaitMsg('Data not entered')
  else begin
    Msg('Please Wait . . .Attribute selection executing. .  ');
    selectWeights(atbUsed);
```

```
  end; {if entered}
end;   { select }


{when integer input variables are are known to be between two
predetermined values, this procedure is called.  For example, the number
of alternatives must be between 2 and altmax (currently 200), so limitin
is called with altnum as the variable returned, 200 and 2 as the limits}

procedure limitin(var inval:integer;max,min:integer);
var
  ok : boolean;
begin
  repeat
    read_int(inval,ok);
    if not ok or (inval>max) or (inval<min)
    then begin
      Msg2('Maximum and Minimum values are: ');
      write(max:5,min:5,'  ');
    end;
  until (inval>=min) and (inval<=max);
end;

procedure EditAttributeNames(var NameList: satbvect;
                             var numNames: integer;
                                 maxNames: integer;
                                 NameLength : integer);
const
  lines = 20;
var
  x,y : integer;
  g,h : integer;
  first,last: integer;
  oldFirst, oldLast : integer;
  ch,ch2 : char;
  curr : integer;


procedure displaylist(here, there : integer);
var g : integer;
begin
  clrscr;
  gotoXY(1,1);
  for g := here to there do
  begin
    writeln(NameList[g]);
  end;
end;

procedure boxrange(var top, bottom: integer);
begin
  if top  < 1 then top  := 1;
```

```
  bottom := top  + lines - 1;
  if bottom > numNames
  then begin
    bottom := numNames;
    top    := bottom - lines +1;
    end;
  if top  < 1 then top  := 1;
end;

procedure cursor_range;
begin
  if y <  1         then begin y := 1;     first := first - 1; end;
  if y >  lines     then begin y := lines; first := first + 1; end;
  if y >  numNames  then       y := numNames;
end;


procedure echo;
begin
  cursor_range;
  boxrange(first, last);
  if not (oldFirst = first)
  then begin
    displaylist(first,last);
    oldFirst := first;
  end;
  curr := y + first - 1;
end;

procedure goto_box_pos;
var
  pos : integer;
begin
  Msg2('Goto Alternative Number: ');
  limitin(pos,numNames,1);
  Msg2('');
  first := pos;
  echo;
  y := pos - first + 1;
end;

procedure deleteOption(d:integer; var num: integer);
var h,j : integer;
begin
  for h := d to num do NameList[h] := NameList[h+1];
  num := num - 1;
  oldFirst := 0;
  echo;

end;

begin
```

```
    oldFirst := 0;
    first := 1;
    y := 1;
    x := 1;
    echo;
    repeat

InputStr(NameList[curr],NameLength,x,y,[^E,^X,^M,^Y,^R,^C,^A],ch,G_InsertMode
On);
      case ch of
        ^E : y := y-1;
        ^X : y := y+1;
        ^M : y := y+1;
        ^Y : deleteOption(curr,numNames);
        ^A : goto_box_pos;
        ^R : first := first - 19;
        ^C : first := first + 19;
      end;
      echo;
    gotoXY(x,y);
    until (ch = ^M) or (numNames = 0);
    enteredAttributes := (numNames <> 0);
end;

procedure EnterAttributeNames(var NameList: satbvect;
                              var numNames: integer;
                                  maxNames: integer;
                                  NameLength : integer);
var
  x, y : integer;
  oldNumNames : integer;
procedure displayform;
begin
  clrscr;
  gotoxy(1,1);
  Writeln('Entering Attribute Names');
  Writeln;
  Writeln('Attribute # ');
  Writeln;
  Writeln('Name     :');
end;
procedure setform(num: integer);
begin
  gotoxy(15,3);write(num);
  gotoxy(x,y);
  clreol;
end;
begin
  textcolor(Gl.EditHeadingColor);
  displayform;
  textcolor(Gl.EditItemColor);
```

```
  x := 12;
  y := 5;
  oldNumNames := numNames;
  repeat
    numNames := numNames + 1;
    setform(numNames);
    InputStr(NameList[numNames],NameLength,x,y,[^M],ch,G_InsertModeOn);
  until (NameList[numNames] = '') or ((numNames) >= maxNames);
  if ((numNames) >= maxNames)
  then WaitMsg('Maximum number of names have been entered');
  if (NameList[numNames] = '') then numNames := numNames - 1;
  if numNames <> 0 then enteredattributes := true;
end;


procedure Display;
var
  g,h : integer;
  alldone   : boolean;
  prin_num  : integer;


begin
  OutputDevice(G_CloseOdev);
  if G1.prcom
  then begin
    write(Odev,#15 );
    writeln('this is true');
  end;
  alldone := false;
  prin_num := 1;
  clrscr;
  gotoxy(1,1);
  textcolor(G1.SubMenuHeadingColor);
  writeln(' Output Menu');
  writeln;
  textcolor(G1.SubMenuItemColor);

  writeln(' Pairwise comparison matrix');
  writeln(' dm Weights');
  writeln(' aTtributes used to determine dominance');
  writeln(' Quit - return to main menu');
  repeat
    lookmenu(4,prin_num,0,2,'PWTQ');
    window(1,10,80,25);
    gotoxy(1,1);
    clrscr;
    case prin_num of
      1 : begin
            Writeln(Odev,'The Eigenvector Pairwise comparisons');
            me1(emat);
          end;
```

```
      2 : begin
            writeln(Ddev,'The DM subjective weights');
            ve2(dmwght);
          end;
      3 : begin
            Writeln(' aTtributes used to determine dominance');
            ve3(atbUsed);
          end;
      4 : alldone:=true;
    end;
  window(1,1,80,25);
  until alldone;
  Flush(Ddev);
  Screen_out(6_CloseDdev);
end;

{The eigenvector editor.  One probably should
re-write the edit routine to allow continuous data entry, and then toss
the matin procedure.  Also use the edit for display instead of having
separate procedures (me1,me2) }


{prints help info on the bottom of the screen.}
procedure edithelp;
begin
  clrscr;
  window(1,19,80,25);
  clrscr;
  textcolor(brown);
  writeln('Use the arrow keys on the          ',  chr(24),'  cursor up
PgUp    window up    ');
  writeln('keypad to move the cursor. The     ',chr(25),'  cursor down
PgDn    windor down  ');
  writeln('numbers on the top row of the      ',chr(26),'  cursor right
ctrl',chr(26),'  window right ');
  writeln('keyboard are for data entry.       ',chr(27),'  cursor left
ctrl',chr(27),'  window left ');
  textcolor(yellow);
  write  ('+','  Edit mode on             )');
  textcolor(Gl.EditCursorOnColor);
  write  ('  <ret>  Enter new value');
  textcolor(Gl.EditCursorOffColor);
  write  ('  <ret>   End edit session');
end;


{version of edit 1 for eigenvector.  Limits are atbnum rather than
altnum }
procedure edit2(var a:pointbmat);
var
  ch  : char;
  vin : real;
```

```
    row : integer;
    col : integer;
    x_top_box : integer;
    y_top_box : integer;
    x_bot_box : integer;
    y_bot_box : integer;
    x_top_old : integer;
    y_top_old : integer;
    x_cur_pos : integer;
    y_cur_pos : integer;
    i,j : integer;
    k,l : integer;
    g,h : integer;
    columns : integer;

 procedure boxrange(var left, right, top, bottom: integer);
 begin
   if top  < 1 then top  := 1;
   if left < 1 then left := 1;
   bottom := top  + lines - 1;
   right  := left + columns - 1;
   if bottom > atbnum
   then begin
     bottom := atbnum;
     top    := bottom - lines +1;
     end;
   if right  > atbnum
   then begin
     right  := atbnum;
     left   := right - columns +1;
     end;
   if top  < 1 then top  := 1;
   if left < 1 then left := 1;
 end;

 procedure cursor_range;
 begin
   if x_cur_pos <  1       then x_cur_pos := 1;
   if x_cur_pos >  columns then x_cur_pos := columns;
   if y_cur_pos <  1       then y_cur_pos := 1;
   if y_cur_pos >  lines   then y_cur_pos := lines;
   if x_cur_pos >  atbnum  then x_cur_pos := atbnum;
   if y_cur_pos >  atbnum  then y_cur_pos := atbnum;
 end;

 procedure me3(a: pointbmat; h,k,i,l : integer);
 var
  g       : integer;
  j       : integer;
 begin

   window(81.fwidth+2,2,80,3);
```

```
    textcolor(Gl.EditNameColor);
    gotoxy(1,1);
    clreol;
    for j:= k to l do write(Odev,attb[j]:Gl.fwidth+1);

    window(1,4,Gl.fwidth+1,18);
    textcolor(Gl.EditNameColor);
    gotoxy(1,1);
    for g:= h to i do begin
      gotoxy(1,g);
      write(Odev,attb[g]:Gl.fwidth);
      end;

    window(Gl.fwidth+2,4,80,18);
    textcolor(Gl.EditItemColor);
    gotoxy(1,1);
    for g:= h to i do begin
      gotoxy(1,g);
      for j:= k to l do write(Odev,a^[g,j]:Gl.fwidth:Gl.decdis,' ');
      if (l-k) < (columns-1) then clreol;

      end;
end;

procedure bigcursor(color:integer);
begin
  gotoxy((x_cur_pos-1) * (Gl.fwidth+1) + 1, y_cur_pos);
  textbackground(color);
  write(a^[y_top_box+y_cur_pos-1,x_top_box+x_cur_pos-
1]:Gl.fwidth:Gl.decdis,' ');
  gotoxy((x_cur_pos-1) * (Gl.fwidth+1) + 1, y_cur_pos);
end;

procedure change;
var
  ok        : boolean;
begin
  if (x_top_box+x_cur_pos-1) = (y_top_box+y_cur_pos-1)
  then begin
    window(1,1,80,25);
    Msg2('Diagonal of Matrix must be 1');
    window(Gl.fwidth+2,4,80,20);
  end
  else begin
    bigcursor(Gl.EditCursorOnColor);
    vin := 0;
    read_real(vin,ok);
    if (vin <> 0) and ok
    then begin
      a^[y_top_box+y_cur_pos-1,x_top_box+x_cur_pos-1] := vin;
      a^[x_top_box+x_cur_pos-1,y_top_box+y_cur_pos-1] := 1/vin;
    end;
```

```
    end;
  end;

  procedure echo;
  begin
    boxrange(x_top_box, x_bot_box, y_top_box, y_bot_box);
    if not ((x_top_old = x_top_box) and (y_top_old = y_top_box))
    then begin
      me3(a,y_top_box, x_top_box, y_bot_box, x_bot_box);
      x_top_old := x_top_box;
      y_top_old := y_top_box;
    end;
  end;

  begin
    columns := (80-(G1.fwidth+2)) div (G1.fwidth+1);
    edithelp;
    x_top_box := 1;
    y_top_box := 1;
    x_cur_pos := 1;
    y_cur_pos := 1;
    x_top_old := 0;
    y_top_old := 0;
    echo;
    repeat
      bigcursor(G1.EditCursorOffColor);
      read(Kbd,ch);
      if ord(ch)=27 then read(Kbd,ch);
      ch:=upcase(ch);
      bigcursor(black);
      case ch of
        'H': begin  y_cur_pos := y_cur_pos  -1;cursor_range;end;
        'P': begin  y_cur_pos := y_cur_pos  +1;cursor_range;end;
        'K': begin  x_cur_pos := x_cur_pos  -1;cursor_range;end;
        'M': begin  x_cur_pos := x_cur_pos  +1;cursor_range;end;
        'S': begin  x_top_box := x_top_box  -(columns-1);echo;end;
        'T': begin  x_top_box := x_top_box  +(columns-1);echo;end;
        'I': begin  y_top_box := y_top_box -10;echo;end;
        'Q': begin  y_top_box := y_top_box +10;echo;end;
        '+': change;
      end;
    until Upcase(ch) in [^M];
    window(1,1,80,25);
    gotoxy(1,1);
  end;

  {initial keyboard entry of the eigenvector matrix. Uses pairwise
  comparison method. }
  procedure eigein(var inp:pointbmat;  isize:integer; tags: satbvect);
  var
    g,h : integer;
    bc1,bc2,fc1,fc2 : integer;
```

```
{controls selection of which attribute is more important, and how much}
procedure query(g,h:integer);
var
  ix   : integer;
  flip : boolean;
  temp : real;
  itmp : real;

{Changes the background color of the two items in the pairwise
comparison}
procedure flipcolor(var a,b:integer);
var
  e : integer;
begin
  e := a;
  a := b;
  b := e;
end;

begin
  flip := false;
  Textbackground(hlack);
  gotoxy(10,2);
  writeln('Attribute      ');
  repeat
    fc1 := yellow;
    fc2 := yellow;
    bc1 := black;
    bc2 := Magenta;
    gotoxy(5,3);
    if flip then flipcolor(bc1,bc2);
    Textbackground(bc2);
    Write(tags[g]:10);
    Textbackground(black);
    if flip then write(' < ') else write(' > ');
    Textbackground(bc1);
    write(tags[h]:10);
    Textbackground(black);
    gotoxy(16,3);
    read(Kbd,ch);
    if keypressed then read(Kbd,ch);
    case ch of
    ',','K': flip:=false;
    '.','M': flip:=true;
    end;
  until upcase(ch) in [^M,'P'];
  ix:=1;
  lookmenu(9,ix,2,3,'123456789');
  itmp:=ix;
  if flip then itmp:=1/itmp;
  inp^[g,h]:=itmp;
```

```
  inp^[h,g]:=1/itmp;
end;

begin  {the eigenvector menu }
  clrscr;
  writeln('                THE SCALE AND ITS DESCRIPTION');
  writeln;
  writeln;
  writeln('     1 . . Equal Importance');
  writeln('     2');
  writeln('     3 . . Weak Importance');
  writeln('     4');
  writeln('     5 . . Essential Importance');
  writeln('     6');
  writeln('     7 . . Demonstrated Importance');
  writeln('     8');
  writeln('     9 . . Absolute Importance');
  {read the eigenvector }
  for g:=1 to isize do
    for h:= 1 to isize do
      if g>h
      then inp^[g,h]:=1/inp^[h,g]
      else if h=g
           then inp^[g,h]:=1
           else query(g,h);
end;



{Decision Maker Weight Input. Direct or Sattys Eigenvector Method}
procedure dmwghtin;
var
  vin     : real;
  sumwght : real;
  h       : integer;
  ok      : boolean;

{calculates the DM weights from the pairwise comparicon matrix}
procedure calceigen( inp:pointbmat; isize:integer; var ray:atbvect ;var
con:real);
var
  csum,
  r1sum,
  r2sum,
  r3sum  : atbvect;
  sum,
  lambda : real;
  c      : pointbmat;
  g,h    : integer;

{sums the columns of the eigenvector }
procedure mcsum(mat:pointbmat;var cvec:atbvect; size:integer);
```

```
  begin
    for g:= 1 to size do begin
      cvec[g]:=0;
      for h:= 1 to size do begin
        cvec[g]:=cvec[g]+mat^[h,g];
      end;
    end;
  end; {proc mcsum}


{sums the rows of the eigenvector }
procedure mrsum(mat:pointbmat;var cvec:atbvect; size:integer);
  begin
    for g:= 1 to size do begin
      cvec[g]:=0;
      for h:= 1 to size do begin
        cvec[g]:=cvec[g]+mat^[g,h];
      end;
    end;
  end; {proc mcsum}

{calculates the eigenvector. simplified form.}
begin  {calceigen }
    new(c);
    mcsum(inp,csum,isize);
    for g:=1 to isize do for h:=1 to isize do c^[g,h]:=inp^[g,h]/csum[h];
    mrsum(c,r1sum,isize);
    for g:=1 to isize do r1sum[g]:=r1sum[g]/isize;
    for g:=1 to isize do for h:=1 to isize do c^[g,h]:=inp^[g,h]*r1sum[h];
    mrsum(c,r2sum,isize);
    for g:=1 to isize do r3sum[g]:=r2sum[g]/r1sum[g];
    sum:=0;
    for g:=1 to isize do sum:=sum+r3sum[g];
    lambda:=sum/isize;
    Con:=(sum/isize-isize)/(isize-1);
    ray:=r1sum;
    dispose(c);
end;  {calceigen }

begin  {dmwghtin}
  Msg('Eigenvector Method or Direct entry of weights? (E/D)');
  repeat Read(Kbd,Ch) until Upcase(Ch) in ['E','D'];
  Writeln(Upcase(Ch));
  if upcase(ch)='E'
  then repeat
    if emat^[1,1]=0
    then eigein(emat,atbnum,attb);
    edit2(emat);
    calceigen(emat,atbnum,dmwght,consty);
    if (atbnum >2) and (atbnum < 10)
    then begin
      consty:=consty/cr[atbnum];
```

```
      Msg('Matrix has a consistency of ');
      Write(consty:5:3);
      end
    else Msg('Cannot determine consistency for matrix. May be too large.');
    Write('.  Accept? Y/N ');
    repeat Read(Kbd,Ch) until Upcase(Ch) in ['Y','N'];
  until Upcase(Ch) in ['Y',^M]
  else repeat
    clrscr;
    for h:=1 to atbnum do begin                                    {read DM
weights}
      Write('Enter Decision Makers weighting for attribute ');
      write(attb[h]:61.fwidth,' [',dmwght[h]:61.fwidth:61.decprt,'] ');
      read_real(vin,ok);
      Writeln;
      if ok then dmwght[h] := vin;
    end;
    sumwght:=0;
    for h:=1 to atbnum do sumwght:=sumwght+dmwght[h];
    if sumwght = 0
    then for h:= 1 to atbnum do dmwght[h]:=1/atbnum
    else if abs(sumwght-1.0) > 0.01
         then begin
           for h:=1 to atbnum do dmwght[h]:=dmwght[h]/sumwght;
           writeln;writeln(' Weights have been normalized to sum to one.');
         end;
    Msg('Are these values correct? (Y/N)');
    repeat Read(Kbd,Ch) until Upcase(Ch) in ['Y','N'];
    Writeln(Upcase(Ch));
  until Upcase(Ch) ='Y';
  enteredWeights := true;
end;

const
  lr1 = 16;

{This procedure handels all keyboard data input.  Data can be keyed in
 to begin a new problem; or data in an existing problem can be edited.}
procedure kbdin;
var
  ix   : integer;
  done : boolean;
  g, h : integer;


{Maximize or Minimize each attribute variable}
procedure maxminin;
var
  tch : char;
begin
repeat
  clrscr;
  for h:=1 to atbnum do begin                                      {Max or minimize}
```

```
      Write('Maximize(Y/N) Attribute #');
      write(h:2,'  ',attb[h]);
      if cost_benefit[h]=1 then tch:='Y' else tch:='N';
      write(' [',tch,']');
      repeat Read(Kbd,Ch) until Upcase(Ch) in ['Y','N',^M];
      Writeln(Upcase(Ch));
      if upcase(ch)='Y' then cost_benefit[h]:=1;
      if upcase(ch)='N' then cost_benefit[h]:=-1;
    end;
    Msg('Are these values correct? (Y/N)');
    repeat Read(kbd,Ch) until Upcase(Ch) in ['Y', 'N'];
    Writeln(Upcase(Ch));
  until Upcase(Ch) ='Y';
  end;

  begin { kbdin }
    ix:=1;
    done:=false;
    repeat
      clrscr;
      gotoxy(1,1);
      textcolor(Gl.SubMenuHeadingColor);
      Writeln(' Edit/Input Data');
      Writeln;
      textcolor(Gl.SubMenuItemColor);
      writeln(' aTtribute names edit or delete');
      writeln(' decision makers Weights');
      writeln(' Cost-benefit');
      writeln(' add attriBute names to existing dataset');
      writeln(' Quit - return to main menu');
      lookmenu(5,ix,0,2,'TWCBQ');
      textcolor(Gl.EditItemColor);
      case ix of
        1 : if EnteredAttributes
            then EditAttributeNames(attb,atbnum,atbmax,natblen)
            else EnterAttributeNames(attb,atbnum,atbmax,natblen);
        2 :dmwghtin;
        3 :maxminin;
        4 :EnterAttributeNames(attb,atbnum,atbmax,natblen);
        5 :done := true;
      end;
    until done;
  end;

  {$I \common\exit.inc}

  {initialize variables for startup of new problem}
  procedure init;
  var
    g,h : integer;
  begin
    for g:= 1 to atbmax do for h:= 1 to atbmax do emat^[g,h]:=0;
```

```
  for g:= 1 to atbmax do attb[g]:=blank15;
  for g:= 1 to atbmax do dmwght[g]:=0;
  for g:= 1 to atbmax do cost_benefit[g]:=1;
  atbnum := 0;
  enteredAttributes := false;
end;

procedure start;
begin
  init;
  Msg(' Keyboard or File input? (K/F)');
  repeat Read(Kbd,Ch) until Upcase(Ch) in ['K','F'];
  Write(Upcase(Ch));
  if UpCase(Ch)='F'
  then load
  else kbdin;
end;

procedure main;
var
  menu_item : integer;
  all_done  : boolean;
  good_eqns : boolean;

procedure nomemory;
begin
  Msg('Not enough memory available to run program.');
  Halt;
end;


begin (* main *)
  {$I-}
  new(emat);
  if IOResult<>0 then nomemory;
  {$I+}
  LoadConfig;
{
  Default;
}
  G_InsertModeOn := true;
  G_CloseOdev    := true;
  all_done       := false;
  menu_item      := 1;
  init;
  screen_out(G_CloseOdev);
  repeat
    clrscr;
    textcolor(G1.MainMenuHeadingColor);
    gotoxy(1,1);
    Writeln('   Main Menu');
    Writeln;
```

```
    textcolor(Gl.MainMenuItemColor);
    writeln(' Load new dataset');
    writeln(' select Attributes');
    writeln(' Display results');
    writeln(' Edit input data');
    writeln(' Save current dataset');
    writeln(' Configure system');
    writeln(' eXit program');
    lookmenu(7,menu_item,0,2,'LRDESCX');
    case menu_item of
      1 :Start;
      2 :select;
      3 :Display;
      4 :Kbdin;
      5 :Save;
      6 :Config;
      7 :exxit(all_done);
    end;
  until all_done;
end;

begin
  main;
end.

File: mdmechos.inc
procedure sortOrder(vec:pointalt; var order:laltVect;length:integer);
var
  jump, m, n : integer;
  temp       : real;
  ltemp      : integer;
  alldone    : boolean;
  temporder  : iAltvect;
begin
  for n:=1 to length do temporder[n] := n;
  jump:=length;
  while jump > 1 do begin
    jump :=jump DIV 2;
    repeat
      alldone:=true;
      for m:=1 to length-jump do begin
        n:= m+jump;
        if vec^[m] < vec^[n]
        then begin
          temp:=vec^[m];
          vec^[m]:=vec^[n];
          vec^[n]:=temp;
          ltemp:=temporder[m];
          temporder[m]:=temporder[n];
          temporder[n]:=ltemp;
          alldone:=false;
        end;
```

```
      end; { for }
    until alldone;
  end; { while }
  for n := 1 to length do  order[temporder[n]] := n;
end; { sort }
{Normal was here}

{  matrix echo -used to print the matrix on the printer and in the
display mode of TOPSIS. This proc is not used in the EDIT mode.  }

procedure me2(var a: amat);
var
 g,h,i,j : integer;
 k,l     : integer;
 pfields : integer;
 preference : laltVect;
begin
  for g:=1 to altnum do value_srt^[g] := value^[g];
  sortOrder(value_srt,preference,altnum);
  pfields:=((G_owidth-naltlen) div G1.fwidth)-3;
  if pfields <= 0 then pfields := 1;
  h:= atbnum div pfields;
  if (atbnum MOD pfields) <> 0 then h:=h+1;
  for g:=1 to h do begin
    k:= pfields*(g-1)+1;
    l:= pfields*g;
    if l > atbnum then l:=atbnum;
    write(Odev,'':naltlen,'   ');
    for i:= k to l do write(Odev,Copy(Concat(attb[i],'
'),1,G1.fwidth),' ');
    writeln(Odev);
    for i:= 1 to altnum do begin
      write(Odev,altn[i]:naltlen);
      write(Odev,value^[i]:G1.fWidth:G1.decdis);
      write(Odev,preference[i]:G1.fWidth);
      write(Odev,'  ');
      for j:= k to l do write(Odev,a[i,j]:G1.fwidth:G1.decdis,' ');
      writeln(Odev);
    end;
    writeln(Odev);
  end;
end;

{matrix echo . eigenvector matrix -used to print the matrix on the
printer and in the display mode of TOPSIS.  This proc is not used in the
EDIT mode.  Two separate procedures are used because the Decision Matrix
and the eigenvector matrix are not shaped the same. }

procedure me1(a: pointbmat);
var
 g,h,i,j : integer;
 k,l     : integer;
```

```
 pfields : integer;
begin
  pfields:=(G_owidth div Gl.fwidth)-2;
  h:= atbnum div pfields;
  if (atbnum MOD pfields) <> 0 then h:=h+1;
  for g:=1 to h do begin
    k:= pfields*(g-1)+1;
    l:= pfields*g;
    if l > atbnum then l:=atbnum;
    write(Odev,'           ');
    for i:= k to l do write(Odev,Copy(Concat(attb[i],'
'),1,Gl.fwidth),' ');
    writeln(Odev);
    for i:= 1 to atbnum do begin
      write(Odev,Copy(Concat(attb[i],'       '),1,Gl.fwidth),' ');
      for j:= k to l do write(Odev,a^[i,j]:Gl.fwidth:Gl.decdis,' ');
      writeln(Odev);
    end;
    writeln(Odev);
  end;
end;


{vector echo . Echos vectors to printer or screen.  Again,
two similar procedures because of dis-similar shape}

procedure vel(var a: pointalt;var baltn:saltvect);
var
 g,h,i,j : integer;
 k,l     : integer;
 pfields : integer;
begin
{
  pfields:=(G_owidth div naltlen);
  pfields := 1;
  h:= altnum div pfields;
  if (altnum MOD pfields) <> 0 then h:=h+1;
  for g:=1 to h do begin
    k:= pfields*(g-1)+1;
    l:= pfields*g;
    if l > altnum then l:=altnum;
     for i:= k to l do write(Odev,Copy(Concat(baltn[i]'
'),1,Gl.fwidth),' ');
    writeln(Odev);
    for i:= k to l do write(Odev,a^[i]:Gl.fwidth:Gl.decdis,' ');
    writeln(Odev);
    writeln(Odev);
  end;
}
  for i:=1 to altnum do
    writeln(Odev,baltn[i]:naltlen,' ',a^[i]:Gl.fwidth:Gl.decdis);
    writeln(Odev);
```

```
    writeln(Odev);
    writeln(Odev);
end;

{vector echo. eigenvector }
procedure ve2(var a: atbvect);
var
 g,h,i,j : integer;
 k,l     : integer;
 pfields : integer;
begin
  pfields:=(G_owidth div Gl.fwidth)-2;
  h:= atbnum div pfields;
  if (atbnum MOD pfields) <> 0 then h:=h+1;
  for g:=1 to h do begin
    k:= pfields*(g-1)+1;
    l:= pfields*g;
    if l > atbnum then l:=atbnum;
    for i:= k to l do write(Odev,Copy(Concat(attb[i],'
'),1,Gl.fwidth),' ');
    writeln(Odev);
    for i:= k to l do write(Odev,a[i]:Gl.fwidth:Gl.decdis,' ');
    writeln(Odev);
    writeln(Odev);
  end;
end;

{This sorts the alternatives in order by closeness to the ideal
solution.  Alternative names are sorted with the closeness measures as a
matched pair.  The algorithm used is the shell-sort.  Not as fast as a
quicksort, but much less code.  Much faster than a bubble sort, with
about the same amount of code. }

procedure sort(vec:pointalt; var names:saltvect;length:integer);
var
  jump, m, n : integer;
  temp       : real;
  alldone    : boolean;
  tempname   : namealt;
begin
  jump:=length;
  while jump > 1 do begin
    jump :=jump DIV 2;
    repeat
      alldone:=true;
      for m:=1 to length-jump do begin
        n:= m+jump;
        if vec^[m] < vec^[n]
        then begin
          temp:=vec^[m];
          vec^[m]:=vec^[n];
          vec^[n]:=temp;
```

```
            tempname:=names[m];
            names[m]:=names[n];
            names[n]:=tempname;
            alldone:=false;
          end;
        end; { for }
      until alldone;
    end; { while }
  end; { sort }

File: mdmedit.inc
{The eigenvector and decision matrix editors.  One probably should
re-write the edit routine to allow continuous data entry, and then toss
the matin procedure.  Also use the edit for display instead of having
separate procedures (me1,me2) }


{prints help info on the bottom of the screen.}
procedure edithelp;
begin
  clrscr;
  window(1,19,80,25);
  clrscr;
  textcolor(brown);
  writeln('Use the arrow keys on the          ',  chr(24),'  cursor up
PgUp     window up     ');
  writeln('keypad to move the cursor. The     ',chr(25),'  cursor down
PgDn     windor down   ');
  writeln('numbers on the top row of the      ',chr(26),'  cursor right
ctrl',chr(26),'  window right ');
  writeln('keyboard are for data entry.       ',chr(27),'  cursor left
ctrl',chr(27),'  window left ');
  textcolor(yellow);
  write  ('+','  Edit mode on             )');
  textcolor(G1.EditCursorOnColor);
  write  ('  <ret>  Enter new value');
  textcolor(G1.EditCursorOffColor);
  write  ('  <ret>   End edit session');
end;

{editor for matrix}
procedure edit1(var a:amat);
var
  ch  : char;
  vin : real;
  row : integer;
  col : integer;
  x_top_box : integer;
  y_top_box : integer;
  x_bot_box : integer;
  y_bot_box : integer;
  x_top_old : integer;
```

```
  y_top_old : integer;
  x_cur_pos : integer;
  y_cur_pos : integer;
  i,j : integer;
  k,l : integer;
  g,h : integer;
  pfields : integer;
  columns : integer;

{On large problems, only a portion of the matrix can be
displayed on the screen at a given time.  The four parameters to
boxrange define which portion of the matrix will be displayed. When
the box is moved the parameters are checked to make sure that they are
still within legal limits.  }

procedure boxrange(var left, right, top, bottom: integer);
begin
  if top  < 1 then top  := 1;
  if left < 1 then left := 1;
  bottom := top  + lines - 1;
  right  := left + columns - 1;
  if bottom > altnum
  then begin
    bottom := altnum;
    top    := bottom - lines +1;
    end;
  if right  > atbnum
  then begin
    right  := atbnum;
    left   := right - columns +1;
    end;
  if top  < 1 then top  := 1;
  if left < 1 then left := 1;
end;

{keeps cursor within legal limits of screen.}

{Echos the protion of the matrix defined by h, i, k, and l (which have
been set by boxrange).  Displays the alternative and attribute names to
the side and top of the matrix }

procedure me3(var a: amat; h,k,i,l : integer);
var
 g       : integer;
 j       : integer;
begin
  {echo attributes at the top of the screen}
  window(naltlen+2,2,80,3);
  textcolor(G1.EditNameColor);
  gotoxy(1,1);
  clreol;
  for j:= k to l do write(Odev,attb[j]:G1.fwidth+1);
```

```
  {echo the alternative names on the left side of the screen}
  window(1,4,naltlen+1,18);
  textcolor(Gl.EditNameColor);
  gotoxy(1,1);
  for g:= h to i do begin
    gotoxy(1,g-h+1);
    write(Odev,altn[g]:naltlen);
    end;

  {echo the defined portion of the matrix }
  window(naltlen+2,4,80,18);
  textcolor(Gl.EditItemColor);
  gotoxy(1,1);
  for g:= h to i do begin
    gotoxy(1,g-h+1);
    for j:= k to l do write(Odev,a[g,j]:Gl.fwidth:Gl.decdis,' ');
    if (l-k) < (columns-1) then clreol;
    end;
end;

{changes the background color of the matrix item pointed to by the
cursor.}
procedure bigcursor(color:integer);
begin
  textbackground(color);
  gotoxy((x_cur_pos-1)*(Gl.fwidth+1) + 1, y_cur_pos);
  write(a[y_top_box+y_cur_pos-1,x_top_box+x_cur_pos-1]:Gl.fwidth:Gl.decdis,'
');
  gotoxy((x_cur_pos-1)*(Gl.fwidth+1) + 1, y_cur_pos);
end;

{after operator has hit a '+' reads a new value for the matrix variable
at the cursor location.}
procedure change;
var
  ok        : boolean;
begin
  bigcursor(Gl.EditCursorOnColor);
  vin := 0;
  read_real(vin,ok);
  if (vin <> 0) and ok
  then a[y_top_box+y_cur_pos-1,x_top_box+x_cur_pos-1] := vin;
end;


{decides whether or not to rewrite the entire matrix. If the box hasn't
changed since it was last displayed, matrix wont be rewritten.  this
keeps the screen from flashing alot when the operator is cursoring
around the screen, and not displaying a newportion of the matrix.}

procedure echo;
```

```
begin
  boxrange(x_top_box, x_bot_box, y_top_box, y_bot_box);
  if not ((x_top_old = x_top_box) and (y_top_old = y_top_box))
  then begin
    me3(a,y_top_box, x_top_box, y_bot_box, x_bot_box);
    x_top_old := x_top_box;
    y_top_old := y_top_box;
  end;
end;

procedure cursor_range;
begin
  if x_cur_pos < 1       then begin x_cur_pos := 1;x_top_box := x_top_box -
1;end;
  if x_cur_pos > columns then begin x_cur_pos := columns;x_top_box :=
x_top_box + 1;end;
  if y_cur_pos < 1       then begin y_cur_pos := 1;y_top_box := y_top_box -
1; end;
  if y_cur_pos > lines   then begin y_cur_pos := lines;y_top_box :=
y_top_box + 1; end;
  if x_cur_pos > atbnum  then x_cur_pos := atbnum;
  if y_cur_pos > altnum  then y_cur_pos := altnum;
  echo;
end;


procedure goto_box_pos;
var
  pos : integer;
begin
  window(1,1,80,25);
  Msg('Goto Alternative Number: ');
  limitin(pos,altnum,1);
  Msg('');
  Msg2('');
  y_top_box := pos;
  window(naltlen+2,4,80,18);
  echo;
  y_cur_pos := pos - y_top_box + 1;
end;

begin  {main body of echo}
  columns := (80-(naltlen+2)) div (Gl.fwidth+1);
  edithelp;                {display help menu}
  x_top_box := 1;          {set initial values}
  y_top_box := 1;
  x_cur_pos := 1;
  y_cur_pos := 1;
  x_top_old := 0;
  y_top_old := 0;
  echo;                    {display screen}
  repeat
```

```
    bigcursor(61.EditCursorOffColor);
    read(Kbd,ch);          {read a character and act on it}
    if ord(ch)=27 then read(Kbd,ch);
    ch:=upcase(ch);
    bigcursor(black);
    case ch of
      'H': begin  y_cur_pos := y_cur_pos  -1;cursor_range;end;
      'P': begin  y_cur_pos := y_cur_pos  +1;cursor_range;end;
      'K': begin  x_cur_pos := x_cur_pos  -1;cursor_range;end;
      'M': begin  x_cur_pos := x_cur_pos  +1;cursor_range;end;
      'S': begin  x_top_box := x_top_box  -(columns-1);echo;end;
      'T': begin  x_top_box := x_top_box  +(columns-1);echo;end;
      'I': begin  y_top_box := y_top_box -10;echo;end;
      'Q': begin  y_top_box := y_top_box +10;echo;end;
      '+': change;
      'G': goto_box_pos;
    end;
  until Upcase(ch) in [^M];
  window(1,1,80,25);                {reset original window}
  gotoxy(1,1);                      {goto corner}
end;                               {bye}


{version of edit 1 for eigenvector.  Limits are atbnum rather than
altnum }
procedure edit2(var a:pointbmat);
var
  ch  : char;
  vin : real;
  row : integer;
  col : integer;
  x_top_box : integer;
  y_top_box : integer;
  x_bot_box : integer;
  y_bot_box : integer;
  x_top_old : integer;
  y_top_old : integer;
  x_cur_pos : integer;
  y_cur_pos : integer;
  i,j : integer;
  k,l : integer;
  g,h : integer;
  columns : integer;

procedure boxrange(var left, right, top, bottom: integer);
begin
  if top  < 1 then top  := 1;
  if left < 1 then left := 1;
  bottom := top  + lines - 1;
  right  := left + columns - 1;
  if bottom > atbnum
  then begin
```

```
      bottom := atbnum;
      top    := bottom - lines +1;
      end;
    if right  > atbnum
    then begin
      right  := atbnum;
      left   := right - columns +1;
      end;
    if top  < 1 then top  := 1;
    if left < 1 then left := 1;
  end;

  procedure cursor_range;
  begin
    if x_cur_pos <  1       then x_cur_pos := 1;
    if x_cur_pos >  columns then x_cur_pos := columns;
    if y_cur_pos <  1       then y_cur_pos := 1;
    if y_cur_pos >  lines   then y_cur_pos := lines;
    if x_cur_pos >  atbnum  then x_cur_pos := atbnum;
    if y_cur_pos >  atbnum  then y_cur_pos := atbnum;
  end;

  procedure me3(a: pointbmat; h,k,i,l : integer);
  var
   g       : integer;
   j       : integer;
  begin

    window(Gl.fwidth+2,2,80,3);
    textcolor(Gl.EditNameColor);
    gotoxy(1,1);
    clreol;
    for j:= k to l do write(Odev,attb[j]:Gl.fwidth+1);

    window(1,4,Gl.fwidth+1,18);
    textcolor(Gl.EditNameColor);
    gotoxy(1,1);
    for g:= h to i do begin
      gotoxy(1,g);
      write(Odev,attb[g]:Gl.fwidth);
      end;

    window(Gl.fwidth+2,4,80,18);
    textcolor(Gl.EditItemColor);
    gotoxy(1,1);
    for g:= h to i do begin
      gotoxy(1,g);
      for j:= k to l do write(Odev,a^[g,j]:Gl.fwidth:Gl.decdis,' ');
      if (l-k) < (columns-1) then clreol;

      end;
  end;
```

```
procedure bigcursor(color:integer);
begin
  gotoxy((x_cur_pos-1) * (Gl.fwidth+1) + 1, y_cur_pos);
  textbackground(color);
  write(a^[y_top_box+y_cur_pos-1,x_top_box+x_cur_pos-
1]:Gl.fwidth:Gl.decdis,' ');
  gotoxy((x_cur_pos-1) * (Gl.fwidth+1) + 1, y_cur_pos);
end;

procedure change;
var
  ok        : boolean;
begin
  if (x_top_box+x_cur_pos-1) = (y_top_box+y_cur_pos-1)
  then begin
    window(1,1,80,25);
    Msg2('Diagonal of Matrix must be 1');
    window(Gl.fwidth+2,4,80,20);
  end
  else begin
    bigcursor(Gl.EditCursorOnColor);
    vin := 0;
    read_real(vin,ok);
    if (vin <> 0) and ok
    then begin
      a^[y_top_box+y_cur_pos-1,x_top_box+x_cur_pos-1] := vin;
      a^[x_top_box+x_cur_pos-1,y_top_box+y_cur_pos-1] := 1/vin;
    end;
  end;
end;

procedure echo;
begin
  boxrange(x_top_box, x_bot_box, y_top_box, y_bot_box);
  if not ((x_top_old = x_top_box) and (y_top_old = y_top_box))
  then begin
    me3(a,y_top_box, x_top_box, y_bot_box, x_bot_box);
    x_top_old := x_top_box;
    y_top_old := y_top_box;
  end;
end;

begin
  columns := (80-(Gl.fwidth+2)) div (Gl.fwidth+1);
  edithelp;
  x_top_box := 1;
  y_top_box := 1;
  x_cur_pos := 1;
  y_cur_pos := 1;
  x_top_old := 0;
  y_top_old := 0;
```

```
    echo;
    repeat
      bigcursor(G].EditCursorOffColor);
      read(Kbd,ch);
      if ord(ch)=27 then read(Kbd,ch);
      ch:=upcase(ch);
      bigcursor(black);
      case ch of
        'H': begin  y_cur_pos := y_cur_pos  -1;cursor_range;end;
        'P': begin  y_cur_pos := y_cur_pos  +1;cursor_range;end;
        'K': begin  x_cur_pos := x_cur_pos  -1;cursor_range;end;
        'M': begin  x_cur_pos := x_cur_pos  +1;cursor_range;end;
        'S': begin  x_top_box := x_top_box  -(columns-1);echo;end;
        'T': begin  x_top_box := x_top_box  +(columns-1);echo;end;
        'I': begin  y_top_box := y_top_box -10;echo;end;
        'Q': begin  y_top_box := y_top_box +10;echo;end;
        '+': change;
      end;
    until Upcase(ch) in [^M];
    window(1,1,80,25);
    gotoxy(1,1);
  end;

  File: mdmfile.inc

  {
  Saves dataset out to file.  If file already exists, prompts with a
  "Replace Yes/No" message.
  }

  procedure Save;
  var
    g,h,i: integer;
    ItemLength : integer;
    ok  : boolean;
  begin
    repeat
      Msg('Save: Enter filename  ');
      GetFileName(Filename,'mdm');
      ok := true;
      if exist(FileName)
      then begin
        Msg2('File already exists. Replace (Y/N) ');
        repeat read(Kbd,ch); ch := Upcase(ch); until ch in ['Y','N'];
        if ch ='Y' then ok := true else ok := false;
        end;
    until ok;
  if FileName<>'' then
    begin
      Assign(mdmfile,FileName);
      Rewrite(mdmfile);
      writeln(mdmfile,altnum:15,atbnum:15);
```

```
    for g:=1 to altnum do
    begin
      for h:=1 to atbnum do
      writeln(mdmfile,decmat^[g,h]:15:10);
    end;
    for g:=1 to atbnum do for h:=1 to atbnum do writeln(mdmfile,emat^[g,h]);
    for h:=1 to atbnum do
    begin
      ItemLength := Length(attb[h]);
      writeln(mdmfile,attb[h]:ItemLength);
    end;
    for h:=1 to atbnum do
    writeln(mdmfile,dmwght[h]:15:10);
    for h:=1 to atbnum do
    writeln(mdmfile,cost_benefit[h]:5);
    for g:=1 to altnum do
    begin
      ItemLength := Length(altn[h]);
      writeln(mdmfile,altn[g]:ItemLength);
    end;

    g := 0; h := 0; i := 0;
    if not enteredAlternativeNames then g := -1;
    if not enteredAttributeNames then h := -1;
    if not entered_DM then i := -1;
    writeln(mdmfile,g:5,h:5,i:5);

    Close(mdmfile);
  end;
end;

{
Loads dataset from a file.  Prompts for a filename.  File has to exists
to be read.
}

procedure Load;
var g,h,i : integer;
begin
  listfiles('*.mdm');
  Msg('Load: Enter filename  ');
  GetFileName(Filename,'mdm');
  if (Filename<>'') then if (not exist(FileName)) then
  repeat
    Msg('File not Found: Enter another filename  ');
    GetFileName(Filename,'mdm');
  until exist(FileName) or (FileName='');
  if FileName<>'' then
  begin
    ClrScr;
    Msg('Please Wait. Reading File...');
    Assign(mdmfile,FileName);
```

```
    Reset(mdmfile);
    read(mdmfile,altnum);
    read(mdmfile,atbnum);
    for g:=1 to altnum do
      for h:=1 to atbnum do read(mdmfile,decmat^[g,h]);
    for g:=1 to atbnum do for h:=1 to atbnum do read(mdmfile,emat^[g,h]);
    readln(mdmfile);
    for g:=1 to atbnum do
      readln(mdmfile,attb[g]);
    for g:=1 to atbnum do
      readln(mdmfile,dmwght[g]);
    for g:=1 to atbnum do
      readln(mdmfile,cost_benefit[g]);
    for g:=1 to altnum do
      readln(mdmfile,altn[g]);

    readln(mdmfile,g,h,i);

    enteredAlternativeNames := (g = 0);
    enteredAttributeNames := (h = 0);
    entered_DM := (i = 0);

    Close(mdmfile);
  end;
end; {of Procedure Load}

File: mdminput.inc
const
  lr1 = 16;

procedure MakeForm;
var
  g : integer;
begin
  clrscr;
  gotoxy(1,1);
  Writeln('Alternative # ');
  Writeln;
  Write('Name     :');
  clreol;
  Writeln;
  Writeln('Attributes: ');
  for g := 1 to atbnum do
  begin
    Write(attb[g]:natblen,' :');
    clreol;
    Writeln;
  end;
end;

procedure clearForm(num: integer;displayname: boolean);
var
```

```
  g : integer;
begin
  gotoxy(15,1);write(num);
  gotoxy(15,3);clreol;if displayname then Write(Altn[num]);
  for g := 1 to atbnum do
  begin
    GotoXY(lr1,g+4);clreol;
  end;
end;

procedure DisplayForm(num : integer);
var
  g : integer;
begin
  for g := 1 to atbnum do
  begin
    GotoXY(lr1-6,g+4);
    Write('[',decmat^[num,g]:Gl.fwidth:Gl.decdis,']');
  end;
end;

procedure readForm(num: integer; readname: boolean);
var
  g : integer;
  temp : real;
  ok, done : boolean;
begin
  done := false;
  if readname
  then begin
    gotoxy(12,3);clreol;
    InputStr(altn[num],naltlen,12,3,[^M],ch,G_InsertModeOn);
    if (altn[num] = '') then done := true;
  end;
  if not done
  then for g := 1 to atbnum do
       begin
         GotoXY(lr1,g+4);
         read_real(temp,ok);
         if ok then decmat^[num,g]:=  temp;
       end;
end;

procedure matin(var List: amat);
var
  g : integer;
begin
    textcolor(Gl.EditHeadingColor);
    gotoxy(1,1);
    clrscr;
    MakeForm;
    textcolor(Gl.EditItemColor);
```

```
    for g := 1 to altnum do
    begin
      ClearForm(g,true);
      ReadForm(g,false);
    end;
    entered_DM := true;
end;

procedure matin2(var List: amat; var curr_num: integer; var name:saltvect);
var
  g : integer;
begin
    textcolor(Gl.EditHeadingColor);
    gotoxy(1,1);
    clrscr;
    MakeForm;
    textcolor(Gl.EditItemColor);
    g := altnum;
    repeat
      g := g + 1;
      ClearForm(g,false);
      ReadForm(g,true);
    until altn[g] = '';
    altnum := g - 1;
    entered_DM := altnum <> 0;
end;


{This procedure handels all keyboard data input.  Data can be keyed in
 to begin a new problem; or data in an existing problem can be edited.}
procedure kbdin;
var
  ix   : integer;
  done : boolean;
  g, h : integer;


{Maximize or Minimize each attribute variable}
procedure maxminin;
var
  tch : char;
begin
repeat
  clrscr;
  for h:=1 to atbnum do begin                                    {Max or minimize}
    Write('Maximize(Y/N) Attribute #');
    write(h:2,'  ',attb[h]);
    if cost_benefit[h]=1 then tch:='Y' else tch:='N';
    write(' [',tch,']');
    repeat Read(Kbd,Ch) until Upcase(Ch) in ['Y','N',^M];
    Writeln(Upcase(Ch));
    if upcase(ch)='Y' then cost_benefit[h]:=1;
```

```
    if upcase(ch)='N' then cost_benefit[h]:=-1;
  end;
  Msg('Are these values correct? (Y/N)');
  repeat Read(Kbd,Ch) until Upcase(Ch) in ['Y','N'];
  Writeln(Upcase(Ch));
until Upcase(Ch) ='Y';
end;

procedure MethodMatin;
begin
  if entered_DM
  then edit1(decmat^)
  else if enteredAttributeNames
       then if enteredAlternativeNames
            then begin
              matin(decmat^);
              entered_DM:=true;
            end
            else begin
              matin2(decmat^,altnum,altn);
              entered_DM:=true;
            end
       else WaitMsg('Alternative and Attribute Names Must be entered
first.');
end;


{Make sure decisiom matrix has been entered.}
procedure return;
begin
  if entered_DM
  then done:=true
  else begin
    Msg('You MUST enter the Decision Matrix. Press any key to continue. . .
');
    Read(Kbd,Ch);
  end;
end;

begin { kbdin }
  ix:=1;
  done:=false;
  repeat
    clrscr;
    gotoxy(1,1);
    textcolor(G1.SubMenuHeadingColor);
    Writeln(' Edit/Input Data');
    Writeln;
    textcolor(G1.SubMenuItemColor);
    writeln(' aLternative names edit or delete');
    writeln(' aTtribute names edit or delete');
    writeln(' decision makers Weights');
    writeln(' Cost-benefit');
    writeln(' Decision matrix');
```

```
    writeln(' add alterNative names to existing dataset');
    writeln(' add attriBute names to existing dataset');
    writeln(' Quit - return to main menu');
    lookmenu(B,ix,0,2,'LTWCDNBQ');
    textcolor(Gl.EditItemColor);
    case ix of
      1 : if EnteredAlternativeNames
          then EditAlternativeNames(altn,altnum,altmax,naltlen)
          else EnterAlternativeNames(altn,altnum,altmax,naltlen);
      2 : if EnteredAttributeNames
          then EditAttributeNames(attb,atbnum,atbmax,natblen)
          else EnterAttributeNames(attb,atbnum,atbmax,natblen);
      3 :dmwghtin;
      4 :maxminin;
      5 :MethodMatin;
      6 :matin2(decmat^,altnum,altn);
      7 :EnterAttributeNames(attb,atbnum,atbmax,natblen);
      B :done := true;
    end;
  until done;
end;

File: mdmprint.inc
procedure Display;
var
  g,h : integer;
  n  : pointamat;            {normalized decision matrix}
  alldone    : boolean;
  prin_num   : integer;


begin
  OutputDevice(G_CloseOdev);
  alldone := false;
  prin_num := 1;
  clrscr;
  gotoxy(1,1);
  textcolor(Gl.SubMenuHeadingColor);
  writeln(' Output Menu');
  writeln;
  textcolor(Gl.SubMenuItemColor);

  writeln(' Decision matrix');
  writeln(' Pairwise comparison matrix');
  writeln(' dm Weights');
  writeln(' Alternative selection values');
  writeln(' Sorted alternative selection values');
  writeln(' Quit - return to main menu');
  repeat
    lookmenu(7,prin_num,0,2,'DPWASQ');
    window(1,10,80,25);
    gotoxy(1,1);
```

```
    clrscr;
    case prin_num of
      1 : begin
            Writeln(Odev,'The Decision Matrix');
            me2(decmat^);
          end;
      2 : begin
            Writeln(Odev,'The Eigenvector Pairwise comparisons');
            me1(emat);
          end;
      3 : begin
            writeln(Odev,'The OM subjective weights');
            ve2(dmwght);
          end;
      5 : begin
            for g:=1 to altnum do value_srt^[g] := value^[g];
            for g:=1 to altnum do altn_srt[g] := altn[g];
            sort(value_srt,altn_srt,altnum);
            Writeln(Odev,'Rank Order of alternatives');
            Writeln(Odev);
            ve1(value_srt,altn_srt);
          end;
      4 : begin
            Writeln(Odev,'Alternative selection values');
            Writeln(Odev);
            ve1(value,altn);
          end;
       6 : alldone:=true;
    end;
  window(1,1,80,25);
  until alldone;
  Flush(Odev);
  Screen_out(G_CloseOdev);
end;

File: mdmwght.inc

{initial keyboard entry of the eigenvector matrix. Uses pairwise
comparison method. }
procedure eigein(var inp:pointbmat;  isize:integer; tags: satbvect);
var
  g,h : integer;
  bc1,bc2,fc1,fc2 : integer;

{controls selection of which attribute is more important, and how much}
procedure query(g,h:integer);
var
  ix   : integer;
  flip : boolean;
  temp : real;
  itmp : real;
```

```
{Changes the background color of the two items in the pairwise
comparison}
procedure flipcolor(var a,b:integer);
var
  e : integer;
begin
  e := a;
  a := b;
  b := e;
end;

begin
  flip := false;
  Textbackground(black);
  gotoxy(10,2);
  writeln('Attribute      ');
  repeat
    fc1 := yellow;
    fc2 := yellow;
    bc1 := black;
    bc2 := Magenta;
    gotoxy(5,3);
    if flip then flipcolor(bc1,bc2);
    Textbackground(bc2);
    Write(tags[g]:10);
    Textbackground(black);
    if flip then write(' < ') else write(' > ');
    Textbackground(bc1);
    write(tags[h]:10);
    Textbackground(black);
    gotoxy(16,3);
    read(Kbd,ch);
    if keypressed then read(Kbd,ch);
    case ch of
    ',','K': flip:=false;
    '.','M': flip:=true;
    end;
  until upcase(ch) in [^M,'P'];
  ix:=1;
  lookmenu(9,ix,2,3,'123456789');
  itmp:=ix;
  if flip then itmp:=1/itmp;
  inp^[g,h]:=itmp;
  inp^[h,g]:=1/itmp;
end;

begin  {the eigenvector menu }
  clrscr;
  writeln('                THE SCALE AND ITS DESCRIPTION');
  writeln;
  writeln;
  writeln('     1 . . Equal Importance');
```

```
  writeln('     2');
  writeln('     3 . . Weak Importance');
  writeln('     4');
  writeln('     5 . . Essential Importance');
  writeln('     6');
  writeln('     7 . . Demonstrated Importance');
  writeln('     8');
  writeln('     9 . . Absolute Importance');
  {read the eigenvector }
  for g:=1 to isize do
    for h:= 1 to isize do
      if g>h
      then inp^[g,h]:=1/inp^[h,g]
      else if h=g
           then inp^[g,h]:=1
           else query(g,h);
end;


{Decision Maker Weight Input. Direct or Sattys Eigenvector Method}
procedure dmwghtin;
var
  vin     : real;
  sumwght : real;
  h       : integer;
  ok      : boolean;

{calculates the DM weights from the pairwise comparicon matrix}
procedure calceigen( inp:pointbmat; isize:integer; var ray:atbvect ;var
con:real);
var
  csum,
  r1sum,
  r2sum,
  r3sum  : atbvect;
  sum,
  lambda : real;
  c      : pointbmat;
  g,h    : integer;

{sums the columns of the eigenvector }
procedure mcsum(mat:pointbmat;var cvec:atbvect; size:integer);
  begin
    for g:= 1 to size do begin
      cvec[g]:=0;
      for h:= 1 to size do begin
        cvec[g]:=cvec[g]+mat^[h,g];
      end;
    end;
  end; {proc mcsum}
```

```
{sums the rows of the eigenvector }
procedure mrsum(mat:pointbmat;var cvec:atbvect; size:integer);
  begin
    for g:= 1 to size do begin
      cvec[g]:=0;
      for h:= 1 to size do begin
        cvec[g]:=cvec[g]+mat^[g,h];
      end;
    end;
  end; {proc mcsum}

{calculates the eigenvector. simplified form.}
begin  {calceigen }
    new(c);
    mcsum(inp,csum,isize);
    for g:=1 to isize do for h:=1 to isize do c^[g,h]:=inp^[g,h]/csum[h];
    mrsum(c,r1sum,isize);
    for g:=1 to isize do r1sum[g]:=r1sum[g]/isize;
    for g:=1 to isize do for h:=1 to isize do c^[g,h]:=inp^[g,h]*r1sum[h];
    mrsum(c,r2sum,isize);
    for g:=1 to isize do r3sum[g]:=r2sum[g]/r1sum[g];
    sum:=0;
    for g:=1 to isize do sum:=sum+r3sum[g];
    lambda:=sum/isize;
    Con:=(sum/isize-isize)/(isize-1);
    ray:=r1sum;
    dispose(c);
end;  {calceigen }

begin  {dmwghtin}
  Msg('Eigenvector Method or Direct entry of weights? (E/D)');
  repeat Read(Kbd,Ch) until Upcase(Ch) in ['E','D'];
  Writeln(Upcase(Ch));
  if upcase(ch)='E'
  then repeat
    if emat^[1,1]=0
    then eigein(emat,atbnum,attb);
    edit2(emat);
    calceigen(emat,atbnum,dmwght,consty);
    if (atbnum >2) and (atbnum < 10)
    then begin
      consty:=consty/cr[atbnum];
      Msg('Matrix has a consistency of ');
      Write(consty:5:3);
      end
    else Msg('Cannot determine consistency for matrix. May be too large.');
    Write('.  Accept? Y/N ');
    repeat Read(Kbd,Ch) until Upcase(Ch) in ['Y','N'];
  until Upcase(Ch) in ['Y',^M]
  else repeat
    clrscr;
```

```
    for h:=1 to atbnum do begin                                    {read DM
weights}
      Write('Enter Decision Makers weighting for attribute');
      write(attb[h]:G1.fwidth,' [',dmwght[h]:G1.fwidth:G1.decprt,'] ');
      read_real(vin,ok);
      Writeln;
      if ok then dmwght[h] := vin;
    end;
    sumwght:=0;
    for h:=1 to atbnum do sumwght:=sumwght+dmwght[h];
    if sumwght = 0
    then for h:= 1 to atbnum do dmwght[h]:=1/atbnum
    else if abs(sumwght-1.0) > 0.01
         then begin
           for h:=1 to atbnum do dmwght[h]:=dmwght[h]/sumwght;
           writeln;writeln(' Weights have been normalized to sum to one.');
         end;
    Msg('Are these values correct? (Y/N)');
    repeat Read(Kbd,Ch) until Upcase(Ch) in ['Y','N'];
    Writeln(Upcase(Ch));
  until Upcase(Ch) ='Y';
end;

File: sawm.pas
{$R+,C-,V-}
{
  This program executes the SAWM algorithm.  This program is menu driven.
  The arrow keys on the numeric keypad will move the blinking cursor to
  the option you desire.  The return key, <ret>, then selects that option.
  The original algorithm was developed by C. L. Hwang and K. Yoon,
  Department of Industrial Engineering Kansas State University.
  The program was written by Dan McAnarney.
}
{ Copyright 1984, 1985   Version 2.04   }

program sawm(input,output);
type
  cray = array[1..9] of real;
const
  altmax    = 1000;        {Maximum number of alternatives }
  atbmax    =   10;        {Maximum number of attributes   }
  G_scwidth =   79;        {Screen width (or printer width }
  naltlen   =   30;        {fieldwidth for strings }
  natblen   =    9;        {fieldwidth for strings }
  lines     =   15;
  blank15   =   '';
  cr : cray =( 0.00, 0.00, 0.58, 0.90, 1.12, 1.24, 1.32, 1.41, 1.49);
type
  charset   = set of char;
  pointamat = ^amat;
  amat      = array [1..altmax,1..atbmax]of real;
  pointBmat = ^amat;
  bmat      = array [1..atbmax,1..atbmax]of real;
```

```
  pointAlt  = ^altvect;
  altvect   = array [1..altmax] of real;
  atbvect   = array [1..atbmax] of real;
  ivect     = array [1..atbmax] of integer;
  iAltVect  = array [1..altmax] of integer;

  namealt   = string[naltlen];
  nameatb   = string[natblen];

  AnyString = string[70];
  String3   = string[3];
  saltvect  = array [1..altmax] of namealt;
  satbvect  = array [1..atbmax] of nameatb;

var
  decmat    : pointamat;           {decision matrix}
  testdecmat: pointAmat; {dummy to test memory}
  emat      : pointBmat;      {eigenvector matrix}
  altnum    : integer;        {number of alternatives}
  atbnum    : integer;        {number of attributes}
  dmwght    : atbvect;        {Decision Makers Weights}
  value     : pointalt;       {relative closeness to ideal soln}
  value_srt : pointalt;       {sorted relative closeness vector}
  altn_srt  : saltvect;       {sorted alternative names }
  altn      : saltvect;       {labels for alternative names}
  attb      : satbvect;       {labels for attribute names }
  cost_benefit : ivect;     { 1-benefit, -1-cost}
  FileName  : string[70];
  Line      : string[100];
  mdmfile   : text;           {output disk file. save data }
  Odev      : text;           {Output device, 'lst:','scn:'}
  consty    : real;           {consistency of the eigenvector}
  G_owidth  : integer;        { output device width}
  G_CloseOdev     : boolean;
  entered_DM      : boolean;
  G_InsertModeOn : boolean;
  enteredAlternativeNames : boolean;
  enteredAttributeNames : boolean;


{$I common\numeric.inc}
{$I common\getfname.inc}
{$I common\menu.inc}
{$I common\config.inc}
{$I common\inpstrng.inc}
{$I common\scrnprtr.inc}
{$I mdmfile.inc}
{$I mdmechos.inc}


{when integer input variables are are known to be between two
predetermined values, this procedure is called.  For example, the number
```

```
of alternatives must be between 2 and altmax (currently 200), so limitin
is called with altnum as the variable returned, 200 and 2 as the limits)

procedure limitin(var inval:integer;max,min:integer);
var
  ok : boolean;
begin
  repeat
    read_int(inval,ok);
    if not ok or (inval>max) or (inval<min)
    then begin
      Msg2('Maximum and Minimum values are: ');
      write(max:5,min:5,'  ');
    end;
  until (inval>=min) and (inval<=max);
end;

{$I namein.inc}

{$I mdmprint.inc }

{$I mdmedit.inc}

{$I mdmwght.inc}

{$I mdminput.inc}

{$I sawm.inc}

{$I common\exit.inc}

{initialize variables for startup of new problem}
procedure init;
var
  g,h : integer;
begin
  for g:= 1 to altmax do for h:= 1 to atbmax do decmat^[g,h]:=0;
  for g:= 1 to atbmax do for h:= 1 to atbmax do emat^[g,h]:=0;
  for g:= 1 to altmax do altn[g]:=blank15;
  for g:= 1 to atbmax do attb[g]:=blank15;
  for g:= 1 to atbmax do dmwght[g]:=0;
  for g:= 1 to altmax do value^[g]:=0;
  for g:= 1 to atbmax do cost_benefit[g]:=1;
  altnum := 0;
  atbnum := 0;
  entered_DM     := false;
  enteredAlternativeNames := false;
  enteredAttributeNames := false;
end;

procedure start;
begin
```

```
  init;
  Msg(' Keyboard or File input? (K/F)');
  repeat Read(Kbd,Ch) until Upcase(Ch) in ['K','F'];
  Write(Upcase(Ch));
  if UpCase(Ch)='F'
  then load
  else kbdin;
end;

{$I common\logosam.inc}

procedure main;
var
  menu_item : integer;
  all_done  : boolean;
  good_eqns : boolean;

procedure nomemory;
begin
  Msg('Not enough memory available to run program.');
  Halt;
end;


begin (* main *)
  {$I-}
  new(decmat);
  if IOResult<>0 then nomemory;
  new(emat);
  if IOResult<>0 then nomemory;
  new(value);
  if IOResult<>0 then nomemory;
  new(value_srt);
  if IOResult<>0 then nomemory;
  new(testDecMat);
  if IOResult<>0 then nomemory;
  dispose(testDecMat);
  {$I+}
  logo1;
  LoadConfig;
{
  Default;
}
  G_InsertModeOn := true;
  G_CloseOdev    := true;
  all_done       := false;
  menu_item      := 1;
  init;
  screen_out(G_CloseOdev);
  repeat
    clrscr;
    textcolor(G1.MainMenuHeadingColor);
```

```
    gotoxy(1,1);
    Writeln('   Main Menu');
    Writeln;
    textcolor(Gl.MainMenuItemColor);
    writeln(' Load new dataset');
    writeln(' Run simple additive weighting method');
    writeln(' Display results');
    writeln(' Edit input data');
    writeln(' Save current dataset');
    writeln(' Configure system');
    writeln(' eXit program');
    lookmenu(7,menu_item,0,2,'LRDESCX');
    case menu_item of
      1 :Start;
      2 :sawm;
      3 :Display;
      4 :Kbdin;
      5 :save;
      6 :Config;
      7 :exxit(all_done);
    end;
  until all_done;
end;

begin
  main;
end.

File: topsis.pas
{$R+,C-,V-}
{
  This program executes the TOPSIS algorithm.  This program is menu driven.
  The arrow keys on the numeric keypad will move the blinking cursor to
  the option you desire.  The return key, <ret>, then selects that option.
  The original algorithm was developed by C. L. Hwang and K. Yoon,
  Department of Industrial Engineering Kansas State University.
  The program was written by Dan McAnarney.
}
{ Copyright 1984, 1985   Version 2.04   }

program topsis(input,output);
type
  cray = array[1..9] of real;
const
  altmax     =  400;          {Maximum number of alternatives }
  atbmax     =   25;          {Maximum number of attributes   }
  G_scwidth  =   80;          {Screen width (or printer width }
  naltlen    =   30;          {fieldwidth for strings }
  natblen    =    9;          {fieldwidth for strings }
  lines      =   15;
  blank15    =   '';
  cr : cray =( 0.00, 0.00, 0.58, 0.90, 1.12, 1.24, 1.32, 1.41, 1.49);
```

```
type
  charset   = set of char;
  pointamat = ^amat;
  amat      = array [1..altmax,1..atbmax]of real;
  pointBmat = ^amat;
  bmat      = array [1..atbmax,1..atbmax]of real;
  pointAlt  = ^altvect;
  altvect   = array [1..altmax] of real;
  atbvect   = array [1..atbmax] of real;
  ivect     = array [1..atbmax] of integer;
  iAltvect  = array [1..altmax] of integer;

  namealt   = string[naltlen];
  nameatb   = string[natblen];

  AnyString = string[70];
  String3   = string[3];
  saltvect  = array [1..altmax] of namealt;
  satbvect  = array [1..atbmax] of nameatb;

var
  decmat     : pointamat;           {decision matrix}
  emat       : pointBmat;       {eigenvector matrix}
  altnum     : integer;         {number of alternatives}
  atbnum     : integer;         {number of attributes}
  dmwght     : atbvect;         {Decision Makers Weights}
  pisoln     : atbvect;         {positive ideal solution }
  nisoln     : atbvect;         {negative Ideal soln }
  spid       : pointalt;         {closeness to ideal soln}
  snid       : pointalt;         {closeness to neg-ideal soln}
  value      : pointalt;         {relative closeness to ideal soln}
  value_srt  : pointalt;         {sorted relative closeness vector}
  altn_srt   : saltvect;        {sorted alternative names }
  altn       : saltvect;        {labels for alternative names}
  attb       : satbvect;        {labels for attribute names }
  cost_benefit : ivect;     { 1-benefit, -1-cost}
  FileName   : string[70];
  Line       : string[100];
  mdmfile    : text;            {output disk file. save data }
  Odev       : text;            {Output device, 'lst:','scn:'}
  consty     : real;            {consistency of the eigenvector}
  G_owidth   : integer;         { output device width}
  G_CloseOdev     : boolean;
  entered_DM      : boolean;
  G_InsertModeOn : boolean;
  enteredAlternativeNames : boolean;
  enteredAttributeNames : boolean;


{$I common\numeric.inc}
{$I common\getfname.inc}
{$I common\menu.inc}
```

```
{$I common\config.inc}
{$I common\inpstrng.inc}
{$I common\scrnprtr.inc}
{$I mdmfile.inc}
{$I mdmechos.inc}


{when integer input variables are are known to be between two
predetermined values, this procedure is called.  For example, the number
of alternatives must be between 2 and altmax (currently 200), so limitin
is called with altnum as the variable returned, 200 and 2 as the limits}

procedure limitin(var inval:integer;max,min:integer);
var
  ok : boolean;
begin
  repeat
    read_int(inval,ok);
    if not ok or (inval>max) or (inval<min)
    then begin
      Msg2('Maximum and Minimum values are: ');
      write(max:5,min:5,'  ');
    end;
  until (inval>=min) and (inval<=max);
end;

{$I namein.inc}

{$I mdmprint.inc }

{$I mdmedit.inc}

{$I mdmwght.inc}

{$I mdminput.inc}

{$I topalg.inc}

{$I common\exit.inc}

{initialize variables for startup of new problem}
procedure init;
var
  g,h : integer;
begin
  for g:= 1 to altmax do for h:= 1 to atbmax do decmat^[g,h]:=0;
  for g:= 1 to atbmax do for h:= 1 to atbmax do emat^[g,h]:=0;
  for g:= 1 to altmax do altn[g]:=blank15;
  for g:= 1 to atbmax do attb[g]:=blank15;
  for g:= 1 to atbmax do dmwght[g]:=0;
  for g:= 1 to atbmax do cost_benefit[g]:=1;
  altnum := 0;
```

```
  atbnum := 0;
  entered_DM     := false;
  enteredAlternativeNames := false;
  enteredAttributeNames := false;
end;

procedure start;
begin
  init;
  Msg(' Keyboard or File input? (K/F)');
  repeat Read(Kbd,Ch) until Upcase(Ch) in ['K','F'];
  Write(Upcase(Ch));
  if UpCase(Ch)='F'
  then load
  else kbdin;
end;

procedure main;
var
  menu_item : integer;
  all_done  : boolean;
  good_eqns : boolean;

procedure nomemory;
begin
  Msg('Not enough memory available to run program.');
  Halt;
end;

begin (* main *)
  {$I-}
  new(decmat);
  if IOResult<>0 then nomemory;
  new(emat);
  if IOResult<>0 then nomemory;
  new(spid);
  if IOResult<>0 then nomemory;
  new(snid);
  if IOResult<>0 then nomemory;
  new(value);
  if IOResult<>0 then nomemory;
  new(value_srt);
  {$I+}
  LoadConfig;
  G_InsertModeOn := true;
  G_CloseOdev    := true;
  all_done       := false;
  menu_item      := 1;
  init;
  screen_out(G_CloseOdev);
  repeat
    clrscr;
```

```
    textcolor(Gl.MainMenuHeadingColor);
    gotoxy(1,1);
    Writeln('   Main Menu');
    Writeln;
    textcolor(Gl.MainMenuItemColor);
    writeln(' Load new dataset');
    writeln(' Run TOPSIS');
    writeln(' Display results');
    writeln(' Edit input data');
    writeln(' Save current dataset');
    writeln(' Configure system');
    writeln(' eXit program');
    lookmenu(7,menu_item,0,2,'LRDESCX');
    case menu_item of
      1 :Start;
      2 :topsis;
      3 :Display;
      4 :Kbdin;
      5 :save;
      6 :Config;
      7 :exxit(all_done);
    end;
  until all_done;
end;

begin
  main;
end.

File: makedm.pas
program randu(input,output);
type
  AnyString = string[70];
  String3 = string[3];
var
  count : integer;
  g,h,i : integer;
  FileName: string[70];
  Line    : string[100];
  Ch      : char;
  eigfile : text;                    {output disk file. save data }
  seed    : integer;
  sumarray: array[1..2,1..9]of integer;
  subscr : integer;
  num    : integer;
  alts, atbs:integer;

procedure GetFileName(var Line: AnyString; FileType:String3);
begin
  Line:='';
  repeat
    Read(Kbd,Ch);
```

```
    if Upcase(Ch) in ['A'..'Z',':','\',^M] then
    begin
      write(Upcase(Ch));
      Line:=Line+Ch;
    end;
  until (Ch=^M) or (length(Line)=10);
  if Ch=^M then Delete(Line,Length(Line),1);
  if Line<>'' then Line:=Line+'.'+FileType;
end;


function  randin(var seed : integer): integer;
const
  increment =  13849;       {13849;}
  multiplier = 25137;           {25137;}
  modulus = 32767;
begin
  seed := (multiplier * seed + increment) div modulus;
  seed := abs(seed);
end;


begin (* main *)
  writeln('maxint= ',maxint);
  randomize;
  writeln('This program generates random decision matrices with');
  writeln('the size determined by the user.');
  write('How many alternatives? ');
  readln(alts);
  write('How many attributes? ');
  readln(atbs);
  writeln('Save: Enter filename  ');
  GetFileName(Filename,'mdm');
  if FileName<>'' then
  begin
    Assign(eigfile,FileName);
    Rewrite(eigfile);
  end;
  count := 0;
  for h:=1 to 2 do for g:= 1 to 9 do sumarray[h,g]:=0;
  subscr:=1;

  writeln(eigfile,alts:15, atbs:15);
  writeln('Building decision matrix');

  for g := 1 to alts do
  begin
    for h := 1 to atbs do
    begin
      if (count mod 50) = 0
      then writeln(count:5,'random number is ',num:8);
      count := count + 1;
```

```
      num:=random(maxint) mod 9 + 1;
      writeln(eigfile,num:15);
      sumarray[subscr,num]:=sumarray[subscr,num]+1;
    end;
  end;

  writeln('Building pairwise comparison matrix');
  for g := 1 to atbs do
  begin
    for h := 1 to atbs do
    begin
      writeln(eigfile,'  1.00');
    end;
  end;

  writeln('Building attribute name list');
  for g := 1 to atbs do
      writeln(eigfile,'x',g);

  writeln('Building benefit-cost lists');
  for g := 1 to atbs do
      writeln(eigfile,'  1.00');

  writeln('Building alternative name list');
  for g := 1 to alts do
  begin
      writeln(eigfile,'a',g);
  end;
  writeln(eigfile,'   0   0   0');

  for g:=1 to 2 do begin
    for h:= 1 to 9 do
      write(' ',sumarray[g,h]:5);
      writeln;
  end;
  Flush(eigfile);
  Close(eigfile);
end.
```

MULTIPLE ATTRIBUTE DECISION MAKING METHODS: A COMPARATIVE STUDY

by

DANIEL K. McANARNEY

B.S. (Electrical Engg), Kansas State University, 1981

AN ABSTRACT OF A MASTER'S THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

College of Engineering
Department of Industrial Engineering
Kansas State University
Manhattan, Kansas

1987

ABSTRACT

The objectives of this study were twofold: to determine the suitability of specific MADM methods, and to develop computer programs implementing them in a "user friendly" manner. The four methods main methods were: simple additive weighting, dominance, ELECTRE and TOPSIS. Two attribute methods are used: the eigenvector method, and a newly developed method of multiple level attribute and weight generation. These methods are discussed as are several possible useful interrelationships with the four main methods.

The methods are all compared for their usefulness on a large randomly generated problem. The conclusion of this work is that TOPSIS seems to be the best MADM method in many circumstances.